Prototyping, Domain Specific Language, and Testing

Liguo Yu

Abstract—**Prototyping is a technique widely used in many engineering fields. However, in software engineering, its usage is limited to requirement elicitation. Little research has been done to extend prototyping to other software development activities. In this paper, we present a prototyping-based testing model and describe how to apply prototyping to the testing activities in the entire software development process. In this model, testing of the product is performed against the prototype in every phase of software development. This prototyping-based testing model is then used in a case study to show how domain specific languages can be used to support prototyping-based testing.**

Index Terms—**Domain specific language, prototyping, software development, software testing.**

I. Introduction

It is widely agreed on that quality assurance is one of the dominant factors in the determination of the success of the software industry. One such process to support quality assurance is testing. Testing is an activity performed throughout the software development process. It generally includes unit testing, integration testing, system testing, and acceptance testing. Figure 1 shows the V-model [1], a widely used software testing model. In this model, the activities on the left focus on building an increasingly detailed product, whereas the activities on the right focus on testing the product. The solid lines indicate the software development process. The dashed lines denote the testing of the product artifacts against the corresponding description documents.

A prototype is an original type, form, or instance of some thing serving as a typical basis or standard for other things of the same category. It contains the most representative attributes of a category and can accordingly be used as an example of all the members of the same category [2].

Prototyping has been widely used in many engineering fields, such as automobiles, domestic appliances, and consumer electronics [3]. Consider manufacturing a new product. Because in engineering, there is great uncertainty as to whether the new product will actually do what is desired, the new product often has unexpected problems. It is crucial to test the design before manufacture the product. Generally, a prototype is used to test the function of the new design, rather than building the full product, detecting what the problems are, and building another full product, and so on.

In prototyping-based engineering fields, only part, but not all, of the complete product is implemented. This allows engineers to rapidly and inexpensively test the parts of the product that are most likely to have problems. After the problems in the prototype are solved, the full product can be built following the design of the prototype [4] [5].

In software engineering, prototyping is a technique widely used in the early phases of software development [6]. A rapid prototype is a quickly implemented version of the target software that is going to be delivered to the client. A rapid prototype is generally produced in requirement elicitation to verify and validate the user requirement.

Conventionally, there are two approaches to reusing a rapid prototype: it is either discarded early in the software development process or converted into the final product [6]. We call these two approaches the discard approach and the convert approach; they are shown in Figure 2(a) and 2(b).

The discard approach is appropriate if minimal effort is devoted to building the rapid prototype. It is usually adopted if the rapid prototype is used solely to show report and screen formats or to demonstrate the feasibility of the software design. The disadvantage of the discard approach is that the effort devoted to prototyping does not directly contribute to the final product. The convert approach is to refine the rapid prototype with the knowledge built into it and to convert it to the final product. The convert approach is usually expensive, because significant changes to the design and the implementation of the rapid prototype may be needed.

In software engineering, prototyping has not been widely used beyond requirement testing. However, studies have shown that prototyping can be helpful in some specific application domains, such as concurrent systems [7] [8] [9] [10] [11]. In this paper, we extend prototyping to later phases of software testing and present a prototyping-based testing model. This prototyping-based testing model utilizes domain specific languages to achieve objectives in reducing software development costs.

The remainder of this paper is organized as follows. Section 2 introduces the prototyping-based testing model. Section 3 describes the domain-specific languages. Section 4 presents the prototyping-based testing activities. Section 5 contains a case study. Our conclusions are in Section 6.

Manuscript received October 31, 2006.

Liguo Yu is with the Computer Science and Informatics Department, Indiana University South Bend, South Bend, IN 46615 USA (phone: 574-520-5067; fax: 574-520-6589; email: ligyu@iusb.edu).

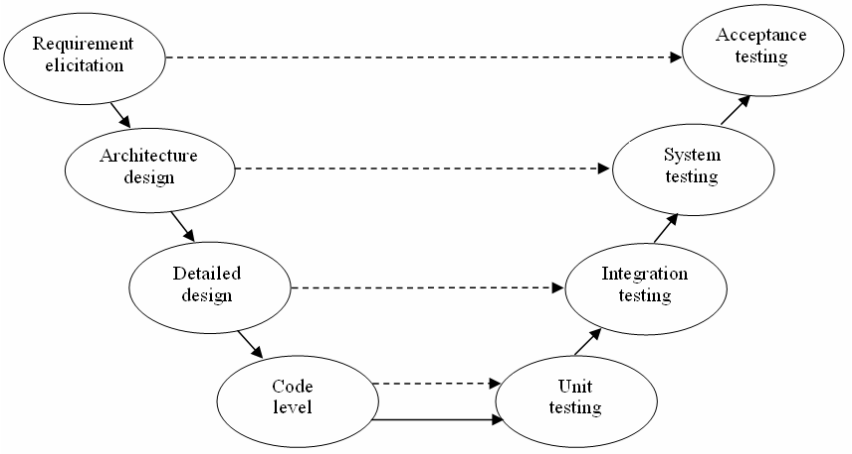

Figure 1. The V-model of software testing [1]

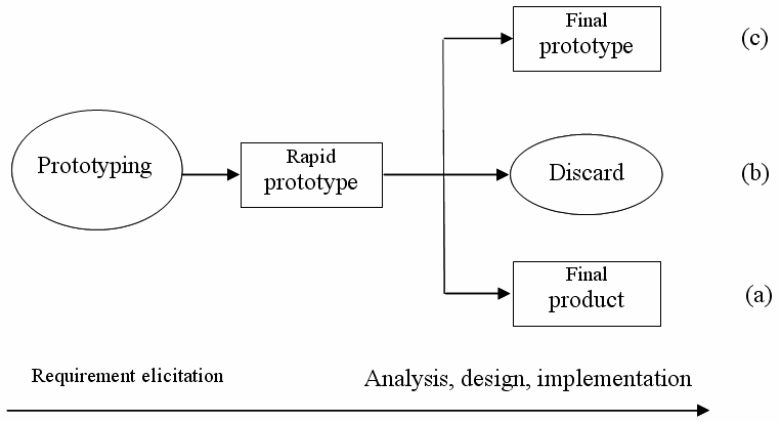

Figure 2. Different approaches to reusing the rapid prototype: (a) convert approach; (b) discard approach; and (c) evolve approach

II. Prototyping-based Testing Model

First, we introduce the concept of final prototype. A *final prototype* is a replica of the final product that will be delivered to the client. This terminology has been widely used in various engineering fields. The final prototype may use different materials and be made with different machines and follow different manufacturing processes. But it functions exactly like a final product, because it conforms to the same design that is used to manufacture the final product. A primary reason to create a final prototype is to insure that all of the parts fit together as planned prior to finalizing production tooling [12].

In this study, the concept of a final prototype is introduced for software engineering. The final prototype is a replica of the target software product. It may be implemented using a different language and run on a different platform, but its architecture and functionalities are identical to those of the final software product.

Next, we introduce a third approach to reusing a rapid prototype, the evolve approach, which is shown in Figure 2(c).

In the *evolve approach*, the rapid prototype is refined to the final prototype in the process of software development. This process is performed in parallel with the development of the final product. Both the final prototype and the final product need to follow the same design specification.

The difference between the evolve approach and convert approach is that in convert approach, the rapid prototype is modified to the final product, whereas in the evolve approach, the rapid prototype is modified to the final prototype. In this study, a prototype is used beyond the early phases of the software process. Therefore, the IEEE definition of prototyping, "*A type of development in which emphasis is placed on developing prototypes early in the development process to permit early feedback and analysis in support of the development process*" [13] is extended to the entire software life cycle.

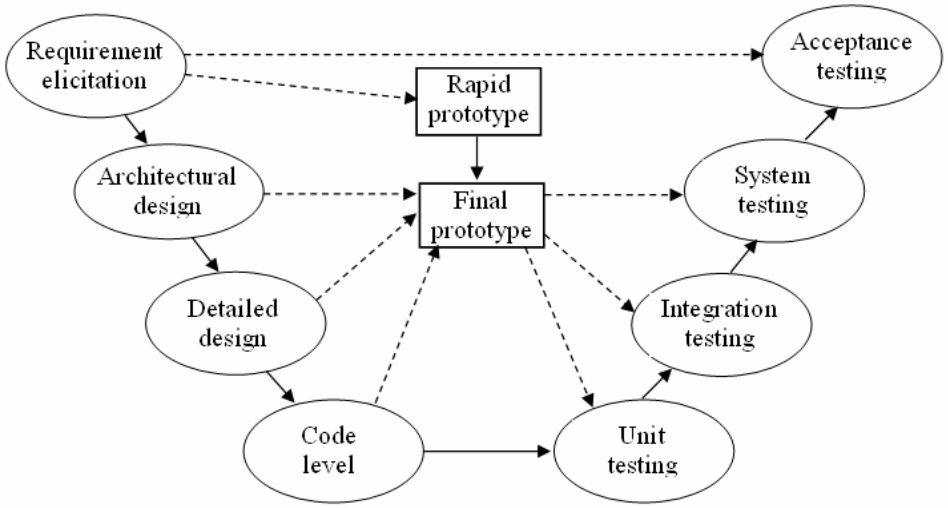

Figure 3. Prototyping-based testing model

Based on the evolve approach, we introduce the prototyping-based testing model, which is shown in Figure 3.

As shown in Figure 3, in this modified V-model, prototyping is performed during the entire software development process, starting with requirements elicitation. The prototype is refined during the architectural design, detailed design, and implementation phases. Testing of the prototype is performed against the corresponding description documents, such as requirement specification, design specification, while testing of the product is performed against the corresponding prototype in every phase of software development.

We remark that (1) the unit testing, integration testing, and system testing are performed against the final prototype. The acceptance testing should be performed again the requirement specification, which is agreed on by both the developers and the clients and is possibly used as the contract; (2) the refinement of rapid prototype to final prototype is an iterative process. There are intermediate prototypes between rapid prototype and final prototype and they are used to test the architecture design and the detail design.

III. Domain Specific Language

The biggest challenge facing prototyping-based testing is cost. Because building a final prototype is time-consuming. The decision as to whether to use this model should be made upon the basis of cost–benefit analysis. Therefore, this model may not be applicable to all software projects. However, the introduction of domain-specific languages in many application areas makes it possible to widely use this model.

A domain-specific language [14] is designed to solve a particular kind of problem, in contrast to general-purpose programming languages. Domain-specific languages can be used to enhance software productivity and reliability in various areas such as graphics, finance, robot control, and so on.

Because a domain-specific language has well defined abstractions and notations, it is more concise and readable. The development time is shortened. Therefore, programming in domain-specific language is much easier than in general-purpose language counterpart.

Domain-specific languages also enable more properties about programs to be checked. Their semantics are restricted to make decidable properties that are critical to a domain. Therefore, testing of a program written in a domain-specific language is much easier than testing the same program written in a general-purpose language.

Besides the advantages, domain-specific languages also have limitations. Due to the predefined formulations, software written in a domain-specific language is less efficient compared to hand-coded software using a general purpose programming language.

Based on these properties, domain-specific languages can be used to support prototyping-based testing in specific domains. For example, consider a software system for breast cancer research. The program first accesses patient database and extracts relevant data. Then it studies the patterns of the data and builds a model to represent the data. Finally, the model is validated and reported to the user. This application belongs to the area of data mining, in which Mathematica [15] is a domain-specific language. The target software needs to process huge amount of data and efficiency is an important issue for this application. Therefore, the final product is required to be coded in Fortran, a general-purpose language to achieve high efficiency. However, the algorithms and models used in this application are complicated and difficult to implement and verify in Fortran. Therefore, Mathematica can be used to implement the prototype and test the product implemented in Fortran.

IV. Prototyping-Based Testing Activities

Now we describe various prototyping-based testing activities. These activities are based on the assumptions that the prototype has been exclusively tested against the specification documents.

Requirement testing: Domain-specific language enables the fast implementation of the rapid prototype. In the early phases of software development, a rapid prototype could be hurriedly built to test the requirement of the customer. After demonstrate the rapid prototype, which includes the key functionalities of the target software, to the customer, the requirement can be finalized.

Design testing: A rapid prototype usually reflects selected functionalities of the product. To continually use prototyping-based testing, it needs to be refined to reflect technical design issues. A domain-specific language enables fast implementation and testing of complicated algorithms, models, and tentative solutions.

Unit testing: The prototype is continually to be refined to the final prototype with the development of the final product. Because the final product and the final prototype follow the same architectural design and detailed design, and some development environments support the integration of domain-specific language with general-purpose language, we can use the units in the final prototype as the testing drivers of the units in the final product. To test a unit of the final product, we replace the corresponding unit in the final prototype. The behavior difference between the two units indicates faults in the unit of the final product.

Integration testing: If the development environment supports the integration of domain-specific language with general-purpose language, it provides more flexible integration approaches for testing the final product, such as a hybrid approach. A set of units of the final product can be integrated gradually in the final prototype environment and tested.

System testing: System testing is used to help identify the correctness, completeness and quality of the entire software system. The same data could be submitted to both the final prototype and the final product. The difference in the output indicates faults in the final product. The final prototype can also be used for comparative testing of the nonfunctional requirements of the final product, such as performance and usability. In this case, the selected nonfunctional requirements should be implemented in both the final prototype and the final product for comparison.

V. Case Study

A. *Description of the Target Software*

The internet has increasingly become a primary source of information for industry, education, and research. Known sites, such as the ACM Digital Library and IEEE, provide rich content that is almost always of some value. However, if one were inclined to search outside domain specific sites using search engine tools such as Google, the results would be far from useful in their raw form. They are littered with blogs, mailing lists, discussion groups, dead links, and spam. Recent technological advances have made it easier for the layman to publish pages filled with speculation, conjecture, and examples that are not representative of any industry standard. A human user often has to click through many useless pages before finding an article of value. On the other hand, the same keyword search might return many different types of useful documents. It is important to categorize the retrieved documents to serve the user needs. Considering the vast amount of documents and library collections, this process is almost impossible for humans to do in a quick and efficient manner. Therefore, it is desirable to have a system which can automatically select and categorize useful documents, and provide valuable information to the user.

Intelligent Document Selector and Categorizer is a semester-long (16 weeks) course project in Software Engineering class at Tennessee Technological University, taken by students majoring in Computer Science. The objective of the project is to produce a C# program under .NET environment so that it can be integrated with an online search engine to select useful documents, extract metadata, and categorize the documents.

Figure 4 shows the data flow of the target program. The students are required to use the pipe-and-filter architectural style for easy integration. The entire data flow can be divided into two processes, document selection process and document categorization process. In the document selection process, the program first uses a web service provided by a search engine to retrieve online documents based on the search criteria. The program then rejects all “obviously bad” documents based on size, link count and other features in the Spam Filter Module. The documents are then rejected / accepted by the predefined rules or blacklist in the Feedback Rejection Module. Finally, the documents are filtered by the specific words frequency in the Supervised Learning Rejection Module and passed to the categorization process.

In the document categorization process, first the Metadata Extraction and Manipulation Module will extract metadata from the documents and divide it into two sets, training data and testing data. The module will then manipulate the training data and use it to train the decision tree. The Data Evaluation Module will test the resulting tree against the testing data set. The results will be printed out to a file and the process will be repeated. Once the process has ended, a human will be able to evaluate the results.

Because data mining techniques are inevitably needed in this application, to help the students understand the problem, an offline Matlab [16] program was provided to show the functions of four modules: Feedback Rejection, Supervised Learning Rejection, Metadata Extraction and Manipulation, and Data Evaluation. The Matlab program takes offline data instead of real-time data as required.

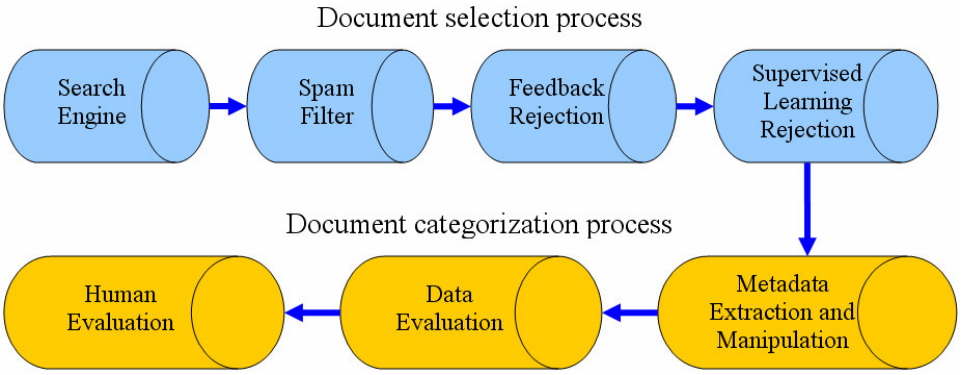

Figure 4. The data flow of the Intelligent Document Selector and Categorizer.

B. Applying Domain Specific Language and Prototyping on Software Development

Two teams took part in this project. Each team contained the same number of programmers. All teams were required to work independently. Teams 1 decided to follow the standard software development life-cycle model as shown in Figure 1. Team 2 decided to follow the prototyping-based development model as shown in Figure 3, that is, they decided to produce a Matlab-based version followed by a C# version. Team 2 chose Matlab because they had been told that Matlab has tool-boxes that could support the fast integration of the software. They made this decision notwithstanding the fact that none of the team members had had any prior Matlab programming experience. The prototyping-based testing is further described below.

Unit testing: Because both the prototype and the product had the same architectural design and because .NET can be integrated with the Matlab program, it was possible to use the modules of the Matlab version as the test drivers of the classes in the C# version. Each time a unit (class) of the C# version was tested, it replaced the corresponding unit (module) in the Matlab version. Differences in behavior between the two units led to the finding of faults in the corresponding unit of the C# version.

Integration testing: Because the Matlab prototype was an integrated system, it provided flexible integrating approaches for testing the C# product. A set of units of Matlab modules was integrated in the .NET environment. Depending on the circumstances, the team used either a top-down approach, a bottom-up approach, a sandwich approach, or a hybrid approach to integrate the C# version.

System testing: System testing tests all the components together. The same data were submitted to both the Matlab prototype and the C# product. Differences in the output indicated faults in the C# product.

C. Results and Experiences

The result of the experiment was that Team 2, which used prototyping-based development, finished the C# version two weeks earlier than Team 1, which had followed the standard software life-cycle model, despite the fact that the members of Team 2 needed additional time to learn Matlab programming first. All two teams passed their acceptance test (product demonstration).

During the course the project, students were required to record their effort (represented as person-hours) spent on the project. Table 1 summarizes the effort of each team. It worth noting that the implementation effort spent by Team 2 contains two parts, the effort to implement the final prototype (32 person-hours) and the effort to implement the final product (24 person-hours). The fourth column shows the percentage effort that is saved by using the prototyping-based development instead of the traditional development. The effort spent on system analysis and system design is considered irrelevant to the testing techniques and the corresponding savings are marked as NA (not applicable). It can be seen that the effort saved on implementation is about 13% and the effort saved on testing is about 17%. The savings of the entire effort is about 5.6%.

Table 1. The effort spent by the two teams (represented in person-hours)

	Team 1	Team 2	Saving
System analysis	44	45	NA
System design	72	68	NA
Implementation	64	56	13%
Testing	18	15	17%

Future systematic experiments and more quantitative data are needed to show that the prototyping-based testing technique is a worthwhile approach. With regard to the restricted experiment we conducted, we believe that the following factors contributed to the success of using this technique:

1. Matlab is a domain-specific language for the application.

It incorporates well-designed tool-boxes that make it easy to implement the prototype. The implementation of the product was easier because of the knowledge gained during the implementation of the prototype;
2. Like most domain specific languages, Matlab is easy to learn;
3. The detailed design contained complicated algorithms that had to be coded, debugged, and fixed many times before they could be implemented correctly. These algorithms were easy to implement and debug in Matlab. Again, the experience gained performing the implementation in Matlab facilitated the subsequent C# implementation; and
4. The .NET development platform supported the integration of the Matlab modules, which made it possible to integrate C# classes and Matlab modules to perform the unit testing and the integration testing.

Our experience is that prototyping-based testing has the following strengths, if used appropriately:
1. It can be used to test the feasibility of the architectural design, detailed design, and complex algorithms;
2. It can obviate the need of test drivers for unit testing;
3. It can provide more flexibility for integration testing; and
4. It can provide comparative system testing. In this project, the Matlab prototype was used for the comparative testing of functional requirements such as the correctness of the C# product. In addition, it could have been used for the comparative testing of the nonfunctional requirements of the C# product, such as performance, security, and usability. The selected nonfunctional requirements could have been implemented in both the Matlab version and the C# version for the purpose of later comparative testing.

On the other hand, our experience shows that prototyping-based testing has the following limitations:
1. Prototyping-based testing cannot be used as the sole testing technique, because the final prototype may not have been implemented correctly. Testing of the final product against the corresponding specifications is also required;
2. Building two versions of a product usually takes longer than just one. The decision as to whether to use prototyping-based testing should be made upon the basis of cost–benefit analysis. This is a project management issue, and involves the nature of the software product to be built, the requirements, the personnel involved, and so on;
3. If the design and implementation are straightforward, the additional overhead of prototyping-based testing may not be needed. Prototyping-based testing is best for complex and mission-critical system development and testing; and
4. For prototyping-based testing, the prototype and the product should be implemented in different languages, preferably a domain specific language (for speed in development) and a general purpose language (for execution speed). If, for some reason, the same language has to be used for both versions, they should be implemented by different teams.

VI. Conclusions

In this paper, we have presented a prototyping-based testing model to be supported by domain-specific languages. This model is shown to be cost efficient in a course project.

Prototyping-based testing can be extended to the entire software-lifecycle including both the development process and the maintenance process. In this case, the final prototype should be updated with the evolution of the final product. Therefore, prototyping can also be used for regression testing in software maintenance.

References

[1] B. Bruegge and A.H.Dutoit, Object-Oriented Software Engineering Using UML, Patterns, and Java, Pearson Prentice Hall, Upper Saddle River, NJ, 2004.
[2] C.K. Chua, K.F. Leong, and C.S. Lim, *Rapid Prototyping; Principles and Applications*, World Scientific, 2003.
[3] K.G. Cooper, *Rapid Prototyping Technology; Selection and Application*, Marcel Dekker Publisher; January 2001.
[4] P.. Hilton and P. Jacobs, eds *Rapid Tooling; Technologies and Industrial Applications*, Marcel Dekker Inc., 2000.
[5] D.T. Pham and S.S. Dimov; *Rapid Manufacturing: the Technologies and Applications of Rapid Prototyping and Rapid Tooling*, Springer Verlag Publishing, January 2001.
[6] F. Kordon and Luqi, "An Introduction to Rapid System Prototyping", *IEEE Transactions on Software Engineering*, vol. 28, no. 9, 2002, pp. 817–821.
[7] W. Hasselbring, "Programming Languages and Systems for Prototyping Concurrent Applications", *ACM Computing Surveys*, vol. 32, no. 1, 2000, pp. 43–79.
[8] B. Meyer, "Systematic Concurrent Object-Oriented Programming", *Communications of the ACM*, vol. 36, no. 9, 1993, pp. 56–80.
[9] J. Schaeffer, D. Szafron, G. Lobe, and I. Parsons, "The Enterprise Model for Developing Distributed Applications", IEEE Parallel Distrib. Technol. Vol. 1, no. , 1993, pp. 85–96.
[10] E. Shapiro, "The Family of Concurrent Logic Programming Languages", ACM Computing Surveys, vol. 21, no. 3, 1989, 413–510.
[11] D.R. Smith, "KIDS: a Semiautomatic Program Development System", IEEE Transactions on Software Engineering, vol. 16, no. 9, 1990, pp. 1024–1043.
[12] Invention City, 2006, http://www.inventioncity.com/prototypes.html
[13] C.J. Booth and G.P. Kurpis, *The new IEEE Standard Dictionary of Electrical and Electronics Terms*, fifth edition, New York: IEEE, 1993.
[14] A.V. Deursen, P. Klint, and J. Visser, "Domain-Specific Languages: an Annotated Bibliography", *ACM SIGPLAN Notices*, vol. 35, no. 6, 2000, pp. 26–36.
[15] Wolfram Research, 2006, http://www.wolfram.com/products-/mathematica/index.html
[16] The MathWorks, 2006, http://www.mathworks.com/

Liguo Yu received the PhD degree in computer science from Vanderbilt University. He is an assistant professor of Computer and Information Sciences Department at Indiana University South Bend. Before joining IUSB, he was a visiting assistant professor at Tennessee Technological University. His research concentrates on software coupling, software maintenance, software reuse, software testing, software management, and open-source software development.