N88-16438

ON ACQUISITION OF PROGRAMMING KNOWLEDGE

Ashok T. Amin
Computer Science Department
The University of Alabama in Huntsville
Huntsville, Alabama 35899

ABSTRACT Acquisition and judicious incorporation of programming knowledge into the programming environment is essential to support development of correct programs and thereby enhance the programmer productivity. A program may be viewed as an outcome of interaction between a programmer and his/her programming environment. The interaction may be supported at the program generation level, program design level, or at the program specification level. The higher is the level of interaction supported the greater is the knowledge base required to support this interaction, and greater are the programming skills required of the programmer. Knowledge acquisition techniques used range from formal-based on mathematical properties of programs to empirical, and from generic to application domain specific. In this paper, we review the recent developments in this area and suggest future directions.

1. INTRODUCTION

Knowledge based programming environments have an important role to play in facilitating the development of correct programs. Such environments can provide for rapid development of correct programs by guiding the programmer through the maize of design decisions and implementation alternatives, and automating low level programming tasks. Developments in this area may lead ultimately to Automatic Programming Systems.

The goal of automatic programming is to automate all the phases of the program development - namely, program specification, design, and implementation. There are two approaches to realization of an Automatic Programming System. In one, the system proceeds with given correct specification of the program and through application of an appropriate sequence of valid transformations transforms the specification into a program. The program specification may be assumed to have been provided by the programmer or developed by the programmer through interactions with the system. As opposed to this, the other approach is more of an evolutionary approach, and is based on extending the automation of programming tasks from lower levels to higher levels. In either case, acquisition and codification of programming knowledge is an essential task. It has been observed [3] that large body of programming knowledge exists and that codification of this knowledge remains most limiting factor to ultimate development of automatic programming systems.

Knowledge acquisition and incorporation for an evolving discipline is more problematic than for a mature one. One finds that programming is variously described as art, craft, and science. In the sense that parts of programming process are at various stages of evolution each of the terms may reasonably be used to describe the programming process. The evolving formalization of programming process by Computer Scientists and the the proven and empirical techniques of programming craft used by the practitioners must be used as the sources of programming knowledge and the knowledge from these source be suitable integrated to realize the gains in automation of programming process.

To further complicate the matter, not only the knowledge bases relevant to various phases of program development need to be integrated, but knowledge bases that address the development of correct programs must be integrated with one that addresses the efficiency concerns in a manner that changes in the one has minimal impact on the other [6]. Often specialization of a programming system to a specific application opens up opportunities for improving the programming process for that application by judicious incorporation of domain specific knowledge [1]. An important aspect of any system is to allow for acquisition and integration of additional knowledge and refinement of existing knowledge during its use by the programmer.

2. PROGRAMMING KNOWLEDGE

Fundamental research in Computer Science deals with formal approaches to understanding of programs and the programming process. The knowledge so gained may be used to develop exploratory techniques for program development. Those techniques that gain acceptance and wide usage then become proven techniques for program development. There are a number of issues that require more empirical approaches for its resolution. Such issues relate to reliability, maintainability, efficiency, human interfaces, etc.

There are two views in relation to approaches to automatic program development. One assumes that formal approaches will ultimately permit automatic development of correct programs. The other view accepts that writing correct programs is hard, and therefore it is accepted that programs may contain errors and that through iterative process of error detection and correction one arrives at a program in which one has high degree of confidence that it is correct. It may be noted that these two approaches complement each other in the sense that formal approaches are necessary if we are to build correct programs for critical applications, and at the same time empirical approaches are necessary if we are to develop any worthwhile program of reasonably large size in timely fashion.

Formal Approaches:

Formal approaches to the understanding of the programs and the programming process demands high degree of profficiency in the creative application of mathematical skills. Formal approaches do not always yield results that may be used to develop practical techniques. Often techniques based on formal approaches are time consuming particularly when it is to be carried out manually. Further, since these techniques require specialized skills, they are subject to subtle and hence hard to detect errors. For example, it may be very difficult to detect error in an erroneous proof of program correctness. None the less knowledge gained from formal inquiries have yielded many useful results.

Laws of programming [5] and formal program design methods [4] are examples of formal approaches to the understanding of programs and the programming process, respectively. It is investigations of these types that will allow us to resolve the questions of program equivalence by suitable representation of programs in a canonical form. And allow development of programs through better understanding of issues that are important at various stages of program design and to the development of languages with suitable degrees of freedom to represent a program through various stages of development.

Exploratory approaches to program development are typically based on some break through in the formal understanding of programs and the programming process. Formal representation of program design information in Programmer's Apprentice represents an exploratory technique [7] for use in semi-automatic program development. Acceptance and wide usage of these techniques leads to proven techniques which then become common knowledge.

Common Knowledge:

Vast body of programming knowledge exists in the form of textbooks and reference books, especially related to algorithms, data structures, and structured programming. The impact of the developments in these areas from formal inquiries in 50's and 60's is now visible in development of programming languages and programmer training. While programming language can not enforce structured programming it can and have facilitated development of structured programs by providing suitable constructs and support for abstract data types. It is believed that advances in programming will require development of languages that support the major paradigms of their user communities [2].

The paradigms for algorithm development, such as divide-and-conquer, provide a systematic approach to development of algorithmic solution to a problem. A unified view of application of this paradigm for the development of algorithms and its implementations for a class of problems, say sorting, is needed to extricate the technique in details enough for the knowledge to be represented in machine processable form [3,8]. To this end, one can recognize that merge sort, quick sort, insertion sort, and selection sort are examples of divide-and-conquer paradigm with differences in the methods of partitioning of the problem and the composition of the solution from the partial solutions. In addition, a number of paradigms need to be developed for the techniques of implementation to be suitably represented, such as recursion-to-iteration transformation, etc.

Empirical Approaches:

There are a number of issues of practical importance that defy formal approaches and in fact may not be properly be subjects of formal inquiry. For example, when do we stop testing a program? A practical answer is when the resources allocated for the testing have been exhausted. None the less, formal inquiries on this questions have yielded results that are awfully inadequate. Other issues involve human interfaces, or measures of complexity. It has been remarked that programming deals with managing complexity. There are rules, based on psychological studies, that say a human can deal with seven things at a time. A number of software methodologies use this as a guide to help manage complexity. How large a problem should be for a technique to more efficient than a simpler one? A number of implementation issues have this characteristics. In such cases it much better to arrive at a resolution based on interaction with the user. The studies involving observations of expert programmer at work also yield useful information involving 'good' programming practices [9].

Acquisition and Incorporation:

The approach used for knowledge acquisition and incorporation is important. Knowledge bases that addresses specific aspects of programming must be integrated in a manner that allows common interface without making it more complex to use or update. The knowledge must be represented so that its refinement and addition can be accommodated gracefully. Lastly, means must be provided so that user may add and modify the programming knowledge.

3. CONCLUSION

For the evolving discipline of programming, acquisition of programming knowledge is a difficult issue. Common knowledge results from the acceptance of proven techniques based on results of formal inquiries into the nature of the programming process. This is a rather slow process. In addition, the vast body of common knowledge needs to explicated to the low enough level of details for it to be represented in the machine processable form. It is felt that this currently impediment to the progress of automatic programming. Importance of formal approaches can not be overestimated since its contributions lead to quantum jump in the state of the art.

REFERENCES

[1] D. Barstow, "Domain-Specific Automatic Programming," IEEE-TSE, 11.11(1985)1321-1336.

[2] R. W. Floyd, "The Paradigms of Programming," Communications

of ACM, 22.8(1979) 455-460.

[3] C. Green and D. Barstow, "On Program Synthesis Knowledge," Artificial Intelligence, 10(1978) 241-279.

[4] C. A. R. Hoare, "An Overview of Some Formal Methods for Program Design," COMPUTER, 20.9(1987) 85-91.

[5] C. A. R. Hoare, I. J. Hayes, et.al, "Laws of Programming," CACM, 30.8 (1987 672-686.

[6] E. Kant, "Efficiency in Program Synthesis" UMI Research Press, 1981.

[7] C. Rich, "A Formal Representation For Plans In The Programmer's Apprentice," Proc. of Seventh International Conference on Artificial Intelligence, 1981.

[8] D. R. Smith, "Top-Down Synthesis of Divide-and-Conquer Algorithms," Artificial Intelligence, 27(1985) 43-96.

[9] E. Soloway and K. Ehrlich, "Empirical Studies of Programming Knowledge," IEEE Trans. of Software Engineering, 10.5(1984) 595-609.