A Rule Based Decision Support System for Programming Language Selection

Meltem Yıldırım İmamoğlu
Department of Computer Engineering
University of Turkish Aeronautical Association
Ankara, Turkey
e-mail: meltem.imamoglu@ceng.thk.edu.tr

Deniz Çetinkaya
Department of Computing and Informatics
Bournemouth University
Poole, United Kingdom
e-mail: dcetinkaya@bournemouth.ac.uk

***Abstract*—One of the most important things in project management is using the most suitable tools and methods in an effective way for project success. The available tools for software project management mainly focus on planning, time management, team management, collaboration, and tracking the development progress. However, there is a lack of supporting mechanisms to guide managers and decision makers for making technical decisions during the early stages. In this paper, we propose a rule based decision support system to guide decision makers and software engineers in programming language selection. Firstly, the system provides a mechanism to build and modify a knowledge base. After that, the system provides guidance about programming language selection before the coding stage according to the project details. The proposed system is implemented with a rule-based programming language.**

Keywords-programming language selection; software engineering; decision support systems; software analysis

I. INTRODUCTION

Information technology field is rapidly growing and changing our daily and business lives. The available tools for software project management such as team management tools, collaborative software development environments, issue trackers, time planning tools, etc. are supporting the development process in a positive way and increasing the project success.

However, there is a lack of supporting mechanisms to guide decision makers for making technical decisions during the early planning and analysis stages. For example, it is not easy to answer the following questions: Which software architecture should be chosen? Which software development tools should be used? Which programming languages should be selected? and so on. These decisions are generally given according to the existing knowhow of the organization or the capabilities of the development team if there is no specific requirement from the customer side about them. In many cases, expert opinion is needed to choose the most suitable and useful options. Specifically, in programming language selection, we try to balance productivity, efficiency, tool support and available resources [1].

In this paper, we propose a rule based decision support system to guide decision makers and software engineers in programming language selection. The users of this system can be technical decision makers, project managers, administrative people, project evaluators, or software engineers, etc. who has knowledge about the project details, but may or may not have knowledge about programming languages. The objective of this work is to provide a mechanism to build and modify a knowledge base about programming languages, as well as, to develop an expert system that works with this knowledge base to provide guidance for programming language selection according to the project details.

The outline of the paper is as follows: Section 1 provides the motivation and objective of this work. Section 2 presents the related work, as well as background information about software engineering and programming languages. Section 3 provides the details of our work, and proposes a rule based expert system for programming language selection. Section 4 explains the prototype implementation of the system. Finally Section 5 draws the conclusions and discusses the future work.

II. RELATED WORK

A rule based decision support system uses rules to build the knowledge base and to do the expert reasoning for solving problems. When we started the literature survey for this work, we were surprised with the limited number of academic resources that presents decision support related studies for programming language selection. Some of them are published more than 20 years ago [2,3]. Although they provide general guidelines that are applicable today, we need new methods since there have been hundreds of new programming languages since then where these languages all have advantages and disadvantages.

Al Ahmar [4] presents the modeling and development of a prototype expert system that helps software project managers and software engineers in selecting the appropriate software development methodology. The paper focuses on the development methodologies and does not discuss about programming languages. Lesani and Rouyendegh [5] present a study that uses fuzzy analytic hierarchy process for the best object-oriented programming language. They apply a limited number of criteria and select from a small set of languages.

There are a couple of studies in the education domain where the focus is which programming language should be taught in the introductory courses [6,7]. Besides, there are expert system proposals in other domains to support the environment and tool selection process [8]. However, they do not provide guidance for programming language selection in software projects. There are also research studies which

978-1-5386-2148-6/17/$31.00 ©2017 IEEE

present comparison of programming languages in the literature [9,10]. They are useful to build the knowledge base, but they do not directly provide a decision support mechanism.

As a result, to the best of our knowledge, there is not any decision support system for programming language selection in software development projects. Yet, the academic studies are limited on this topic and people tend to use the popular languages when there is no constraint for programming language selection.

When we searched the gray literature, we saw that the main concern is which programming language to learn as a first language. Although some big software vendors published reports on programming language selection for projects, they are generally on the level of useful guidelines [11,12]. In many cases, people follow the trend in software development industry and it is generally stated that choosing a programming language is a challenging task.

It was also interesting to see many top ten lists on the Internet for programming languages without formal categorization and any proper data underneath. Some of these lists are prepared according to the open job positions in the industry which is found valuable especially by the new learners but indeed does not provide a professional guideline.

III. A Decision Support System for Programming Language Selection

A programming language is a formal language for writing computer programs or software which are specifications defining a set of instructions that a computer can interpret and produce various kinds of output. The early computer programming languages were in use around early 1950s and today there are hundreds of programming languages at different levels and for different purposes.

Each programming language has some basic building blocks for the description of data and the instructions. These building blocks are defined by the language definition as syntax and semantics of the language. Choosing a programming language to develop computer programs is becoming more and more challenging due to the increasing time and resource constraints in software projects as well as due to the increasing number of available programming languages.

The first thing to check while choosing a programming language is if it is fit-for-purpose. For some specific tasks a domain specific or a specialized programming language can fit better while a general purpose programming language can be more suitable for a standard desktop application. Then, there are other criteria which affect the decision process such as project duration, team skills, organizational constraints, project budget, customer requirements, etc.

In this section, we propose a rule based decision support system to guide decision makers in programming language selection. We first provide a mechanism to build and modify a knowledge base about programming languages. Then we develop an expert system that works with this knowledge base to provide guidance for programming language selection according to the project details. In our system there is a two stages reasoning. First user parameters are checked and some conclusions are drawn and then these conclusions and initial user parameters are used to find the total scores for programming languages. Finally, the results are shown in a sorted list. Fig. 1 shows the general overview of the proposed system.

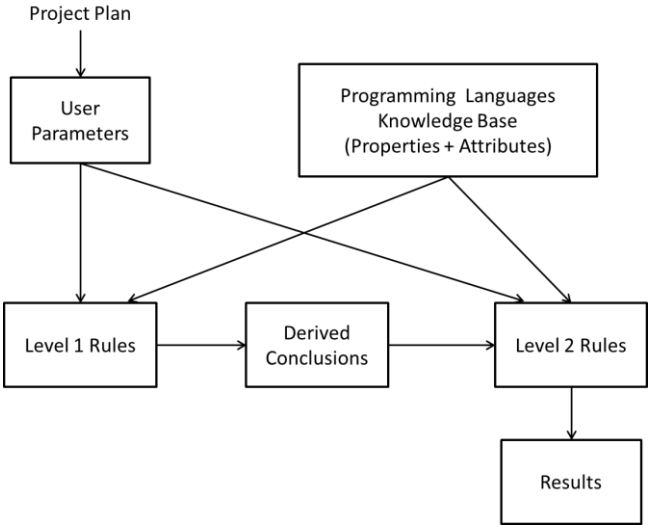

Figure 1. General overview of the proposed expert system.

First of all user defines the parameters according to the project plan and the decisions made at the beginning of the project. User parameters are variables such as project duration, project domain, category, purpose, etc. which are project specific information.

The programming languages knowledge base includes properties and attributes for different programming languages. Each property defines characteristics of a programming language such as its name, category, first appeared year, etc. Each attribute of a language provides more details about the usage of the language such as available tool support, available libraries, documentation support, etc.

Level 1 and Level 2 rules provide a set of conditional rules to derive useful conclusions and to get results to provide project specific guidance based on the user parameters and using the knowledge base. These rules apply weighted relations between the parameters, properties and attributes. If the attribute has a text value, it is checked with if-then rules. If the attribute has a number value, which is in a range of 1 to 10, then it is multiplied with the weight and added to the result.

Table 1 shows a set of sample relations between the user parameters and the language properties and attributes. Table 1 also shows the effect of the parameters on the use of the properties or attributes during scoring as direct or indirect effect. Direct effect means that the parameter is used directly to derive results while indirect effect means that the parameter is first used to derive a conclusion and then the conclusion affects the result. The overall score for each programming language within the suitable category is calculated separately after executing all rules and the ones with the highest scores are shown.

TABLE I. PARAMETERS' RELATION WITH PROPERTIES AND ATTRIBUTES

Parameter name	Parameter effect	Weight	Related property	Related attribute
Category	Direct	0.80	Category	
Category	Direct	0.80		Application platform
Domain	Direct	0.80	Applicable domains	
Duration	Indirect	0.30		Available libraries
Duration	Indirect	0.30		Available IDEs
Language	Direct	0.25		Language support
Number of team members	Direct	0.30	Subcategory	
Number of team members	Indirect	0.30		Available projects
Budget	Direct	0.20		Available free tools
Budget	Direct	0.10		Documentation support
Budget	Direct	0.20		Available sample projects
Budget	Direct	0.20		Available free libraries

IV. IMPLEMENTATION OF THE SYSTEM

A web based expert system for supporting software engineers and software project managers during programming language selection has been developed as a proof of concept, namely PLExpert. The PLExpert system uses CLIPS inference engine with the decisions performed in a forward manner to define knowledge base and to execute the rules. User interfaces are developed with PHP and HTML5.

The user interface allows the user to interact with the system for defining the parameters and getting the results. Fig. 3 shows the web page where the user can define the parameters such as category, domain, project duration, etc.

The user inputs that are required to run the expert system are called as parameters in our study. The parameter template is defined as follows:

```
(deftemplate parameter
  (slot paramname (default none))
  (slot value (default none)))
```

Parameters are transformed to CLIPS rules and plexpert_parameters.bat file is generated. For example, Category parameter can be chosen from different pre-defined enumerated values which can be game, mobile, embedded, mis, expert, scientific, etc. When the user chooses a project category, the following rule is generated:

```
(assert (parameter
   (paramname prj-category) (value game)))
```

This is a well-defined task, and can be automatized easily. In this way, users define the parameters without knowing the details of CLIPS. Undefined user parameters are not included during the execution.

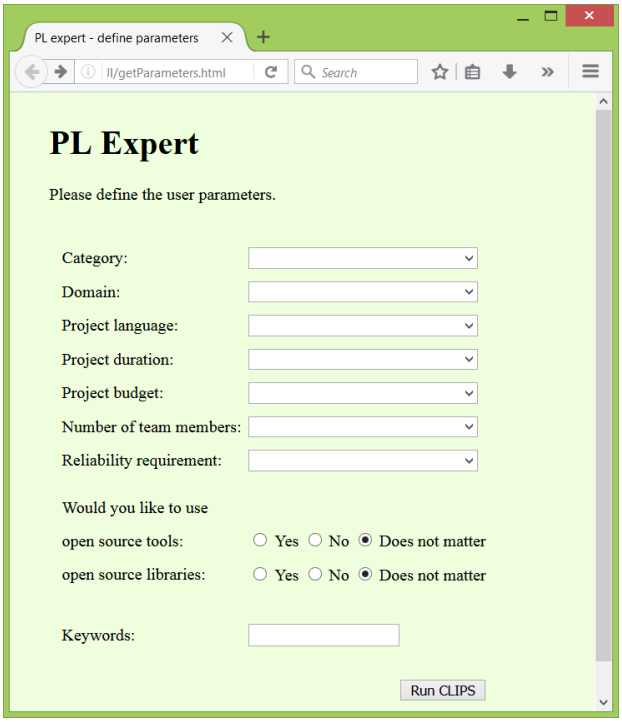

Figure 2. User interface for getting the parameters.

The knowledge base includes the programming language definitions with basic properties and extra attributes that can be associated with the defined languages. The programming language (PL) template and the attribute template are as follows:

```
(deftemplate PL
   (slot langname)
   (slot category)
   (slot subcategory)
   (slot year))

(deftemplate attribute
  (slot attname)
  (slot language)
  (multislot value))
```

The knowledge base is developed by using three most commonly used search engines, Google, Bing and Yahoo, which covers around %90 of the market share. We first searched the programming languages and identified 24 languages to be added in our knowledge base including Java, C, C++, PHP, JSP, Lisp, Prolog, Python, etc. Then we added attributes for each language, such as open source editors, available libraries, tutorial support, etc. We arranged the scores and weights according to the search results and by using expert opinion.

The PLExpert system can be configured and scores and weights can be changed easily. Besides, new languages can be added for the future use of the system. Hence, our objective was not providing the perfect knowledge base during this study but developing a structured mechanism that we can define the programming languages knowledge base.

Expert knowledge is represented in our system with CLIPS rules. For example, the following rule is written to express the expert knowledge "if project complexity is high, then the programming languages which are first appeared more than 15 years ago will have more chance to be successful."

```
(defrule selectRuleComplexity1
    (and (conclusion (name prj-complexity)
(score ?s))
      (PL (langname ?n) (category ?c)
(subcategory ?sc) (year ?y)))
    (test (> ?s 40))
    (test (< ?y 2002))
    =>
    (assert (result (language ?n) (score
5.0))) )
```

We have around 10 rules for deriving conclusions and 20 rules for getting scores. A sample scoring function to derive conclusion about project complexity is given below:

```
   <prj-complexity> = (<prj-duration>/12 +
<prj-team>/5 + <prj-reliability-level>/2)*10
```

Fig. 3 shows the implementation environment with CLIPS and Fig. 4 shows the overall implementation structure.

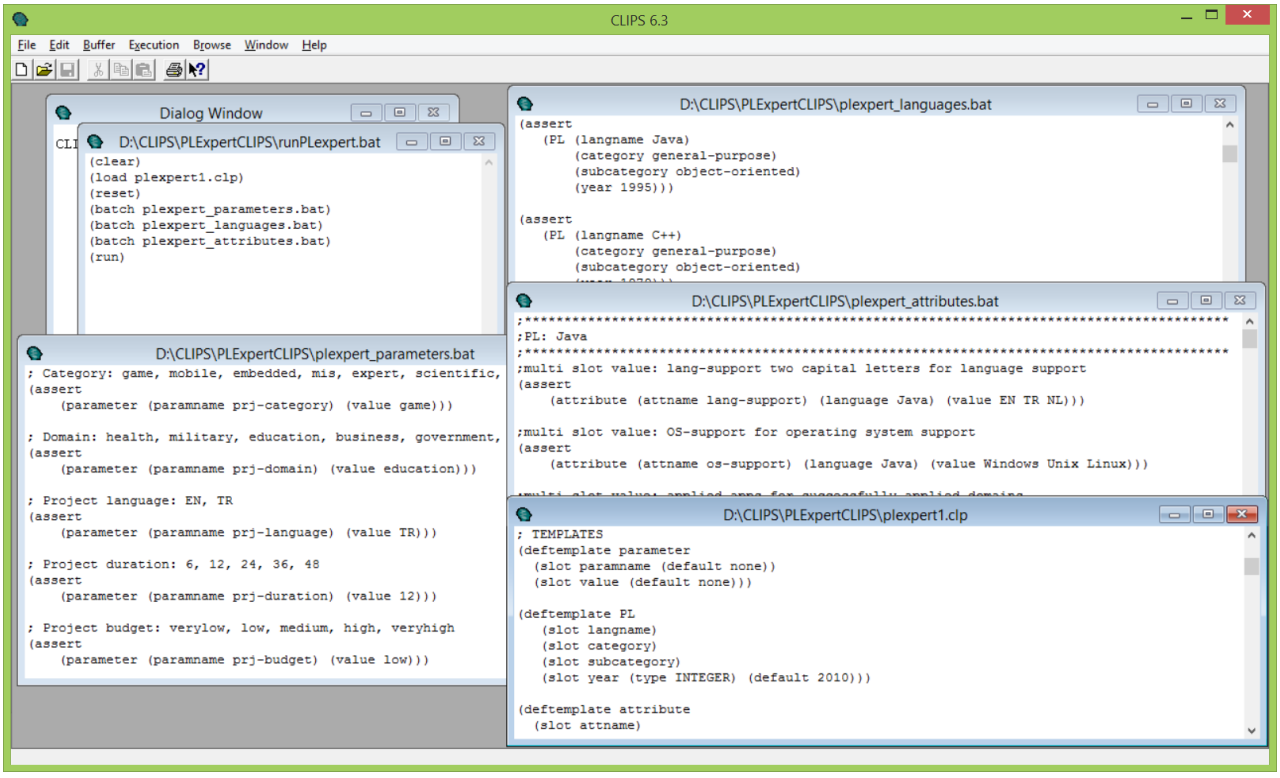

Figure 3. Implementation environment with CLIPS.

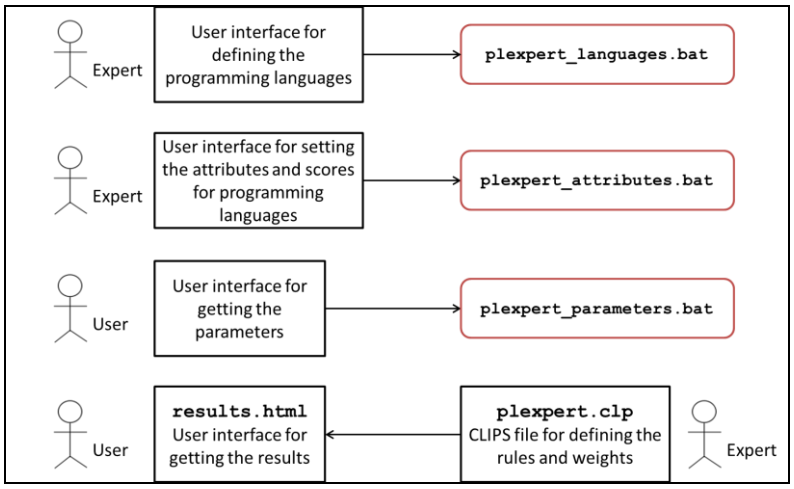

Figure 4. User interfaces and the files in the PLExpert system.

Once we run the PLExpert, we get the results in CLIPS as shown in Fig. 5. Then we present the results in an HTML page with a table.

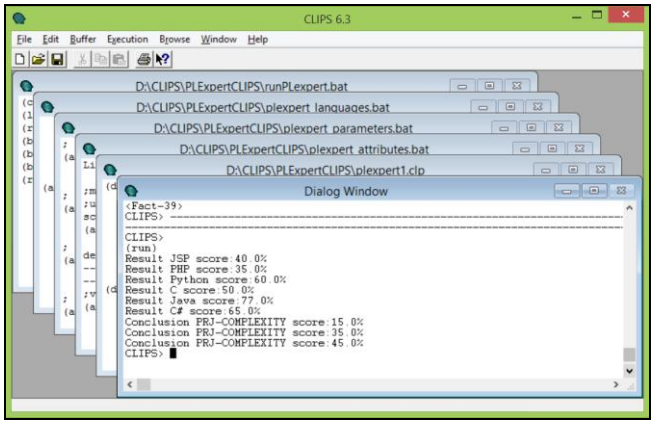

Figure 5. Execution results in CLIPS.

We have tested our system with sample project scenarios. During the evaluation we tested both successful and unsuccessful project cases. For example, we know that MATLAB is not normally suitable for mobile application development since it is for numerical computing. Or, Objective-C is the best language for iOS platforms. However, we need more tests with real life applications and projects to use the proposed system in real life.

V. CONCLUSION

We have developed a rule based expert system for selecting the most suitable programming language for software projects. The proposed expert system is CLIPS-based and the rules are written manually.

As a future work, we would like to auto generate the rules with the help of model driven approaches. In this way, we aim to provide a way to software developers for calibrating the rules according to their specific needs. Besides, we would like to extend our work with adding programming tool and environment selection support. During our tests we recognized that today's applications are often written with multiple compatible languages. Hence, we will improve our system with multiple language suggestions.

REFERENCES

[1] D. Spinellis, "Choosing a programming language," IEEE Software, vol. 23(4), 2006, pp. 62-63. doi: 10.1109/MS.2006.97

[2] C.J. Burgess, "Software quality issues when choosing a programming language," WIT Transactions on Information and Communication Technologies, vol. 14, 1995.

[3] J. Howatt, "A project-based approach to programming language evaluation," SIGPLAN Not., vol. 30(7), July 1995, pp. 37-40. doi: 10.1145/208639.208642

[4] M. A. Al Ahmar, "Rule based expert system for selecting software development methodology," Journal of Theoretical and Applied Information Technology, vol. 19(2) , 2010, pp. 143-148.

[5] S. H. Lesani, and B. D. Rouyendegh (B. Erdebilli), "Object-oriented programming language selection using fuzzy AHP method," Presented at the annual meeting of the ISAHP, 2014.

[6] K. R. Parker, J. T. Chao, T. A. Ottaway, J. Chang, "A formal language selection process for introductory programming courses," Journal of Information Technology Education, vol. 5, 2006.

[7] A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson, "A survey of literature on the teaching of introductory programming," In Working group reports on ITiCSE on Innovation and technology in computer science education, 2007, pp. 204-223. doi: 10.1145/1345443.1345441

[8] N. Vargas Hernandez, G. Okudan Kremer, L.C. Schmidt, and P.R. Acosta Herrera, "Development of an expert system to aid engineers in the selection of design for environment methods and tools," Expert Systems with Applications, vol. 39(10), 2012, pp. 9543-9553.

[9] K. Aldrawiesh and A. Al-Ajlan, "Selecting the best object-oriented programming language for developing distributed computing systems," Proc. of the International Conference on Computer Engineering & Systems, Cairo, 2009, pp. 440-446. doi: 10.1109/ICCES.2009.5383225

[10] M. Fourment and M. R. Gillings, "A comparison of common programming languages used in bioinformatics," BMC Bioinformatics, vol. 9(82), 2008. doi: 10.1186/1471-2105-9-82

[11] C. Britton, "Choosing a programming language," Microsoft MSDN library, available via https://msdn.microsoft.com/en-us/library/cc168615.aspx, 2008.

[12] J. Reghunadh and N. Jain, "Selecting the optimal programming language - Factors to consider," IBM developerWorks, available via https://www.ibm.com/developerworks/library/wa-optimal, 2011.