Become a Star*
Teaching the Process of Design and Implementation of an Intelligent System

{* domain expert, knowledge engineer, programmer, end user, project manager}

Anne Venables and Grace Tan
Victoria University, Melbourne City, Australia

Anne.Venables@vu.edu.au Grace.Tan@vu.edu.au

Abstract

Teaching future knowledge engineers, the necessary skills for designing and implementing intelligent software solutions required by business, industry and research today, is a very tall order. These skills are not easily taught in traditional undergraduate computer science lectures; nor are the practical experiences easily reinforced in laboratory sessions. In an attempt to address this issue, a software development project, designed to take students through a complete process of knowledge engineering, was introduced in an undergraduate Intelligent Systems subject. In this project, students were required to act as domain experts, knowledge engineers, programmers, end users and project manager in the production of a game-playing expert system. The paper describes the project, its objectives and development, as well as some of the benefits.

Keywords: Intelligent Systems, Expert Systems Development, Role-playing Learning, Game Design.

Introduction

Typically Intelligent Systems (IS) or Artificial Intelligence (AI) subjects are offered in the final year of a Computer Science degree, since students need some programming and software development experience to contend with the scope of the subject. The content is broad and ever-changing: spanning a diverse and disparate range of topics from expert systems and robotics, through machine intelligences and neural networks; hinging on the cutting edge of computer sci ence research; and, prominently represented in real-world applications. Such subject offerings should introduce students to different problem solving strategies and heuristics of IS and AI, as well as aim to help students gain an understanding of the development of an IS and the process of knowledge engineering.

Within the tight delivery frameworks and time limits of a typical teaching semester, it is difficult to cover all the different fields of study in an IS subject well. Some lecturers faced with such diverse subject matter, opt to do a minimal subset of topics, choosing one problem-solving strategy and focusing on some of its aspects. As a consequence, the supporting student laboratory sessions center on using artificial practice instances of the chosen strategy. Using this approach, it is difficult for stu-

Editor: Eli Cohen

dents to develop the necessary knowledge engineering skills since they are not given the opportunity to work on a problem of any breath or depth.

As an elective study in a Computer Science degree, the subject of Intelligent Systems mostly attracts students majoring in computer science, however it also appeals to engineering students. All enrolling students must have completed two programming subjects and some will have studied software engineering and/or software development subjects. As a consequence of the liberal prerequisite requirements and broad subject appeal, the subject includes a heterogeneous student group representing a broad range of background skills and differing ways of learning. The varying programming backgrounds of IS students (Hill & Alford, 2004), and availability of suitably equipped laboratories and software, are other limiting factors in the delivery of such subject material. Hence, at no stage do students have an opportunity to experience commercial knowledge engineering software, or develop and create an intelligent system?

Rationale for & Design of the Project

Given the above difficulties of presenting IS subject content, it was decided that one way of addressing those complexities would be through the introduction of a software development project. Even though the task itself is not considered a capstone project for the degree, it would take students through the complete process of knowledge engineering, where there would be as much emphasis on the process of creation as on the end product itself. As noted by Ye and Gray (1996) who were also frustrated by similar teaching challenges in their accounting information course, "teaching expert system development is particularly difficult" in that it requires an understanding of the decision making processes in creating such projects. In our case, the major pedagogical goal was primarily to expose students to the development process and, particularly the various roles of domain expert, knowledge engineer, programmer, end user and project manager that they would need to play throughout, as students could be expected to play any of these roles in their future professional careers.

Such role-playing strategies and exercises as a vehicle for student instruction are reported throughout the computer science education literature, particularly in courses that require students to plan, estimate, organize and communicate (Dean & Hinchey, 1995; Jones, 1987; Sullivan, 1993; Zowghi & Paryani; 2003). This approach emphasizes integration of theory with practice and is associated with the experiential way of learning, described by Kolb (1984). Previously Kolb and Fry (1975) described it as a cyclical process passing through four stages: experiencing, reflecting, concluding and testing. The role-playing strategy was specifically structured to link classroom and textbook theory to practice, in a way that would promote 'deep', rather than 'surface', learning. It requires that students apply their programming skills and the theoretical knowledge of problem-solving strategies and heuristics to the development of a rule-based expert system. By taking students through the complete process of knowledge engineering, such problem-based learning encourages the simultaneous development of a knowledge base and skills (Soh, 2004) together with a capacity for independent work. Hopefully, such a strategy would encourage a development of the craft of knowledge engineering (Armarego & Clarke, 2003) whilst equipping students with the expertise and enough practical experience to approach future problems in their working careers.

The project involved the creation of an expert system capable of playing a game. An expert system is "a computer program that represents and reasons with knowledge of some specialist subject with a view to solving problems or giving advice" (Jackson, 1999). For instance, an often cited historical example is MYCIN which was a software developed by Stanford University in the 1970s. In testing, MYCIN could diagnose infectious blood diseases to the same ability as human expects and it performed better than novice doctors during testing (Cawsey, 1994). Today, expert systems are endemic in business and research. They perform a myriad of tasks including data in-

terpretation, such as sonar signals, diagnosis of malfunctions in equipment, analysis of complex chemicals, configuring of computer systems and planning of sequential actions of robots (Jackson, 1999).

Typically, a real-world expert system development is conducted by a team of various skilled members, as shown in the Figure 1 (Negnevitsky, 2002). Under the direction of the project manager, it is the primary task of the knowledge engineer to question and to elicit the expert knowledge residing with the domain expert, with the goal of encapsulating that knowledge in a set of rules, or heuristics. Subsequently, these rules are encoded, by the programmer, in an intelligent system. An end user comes to this program seeking expert advice equivalent to that given by the domain expert, without the step of consulting the human expert directly. Thus the machine or artificial "intelligence" appears to the end-user when they query the expert system.

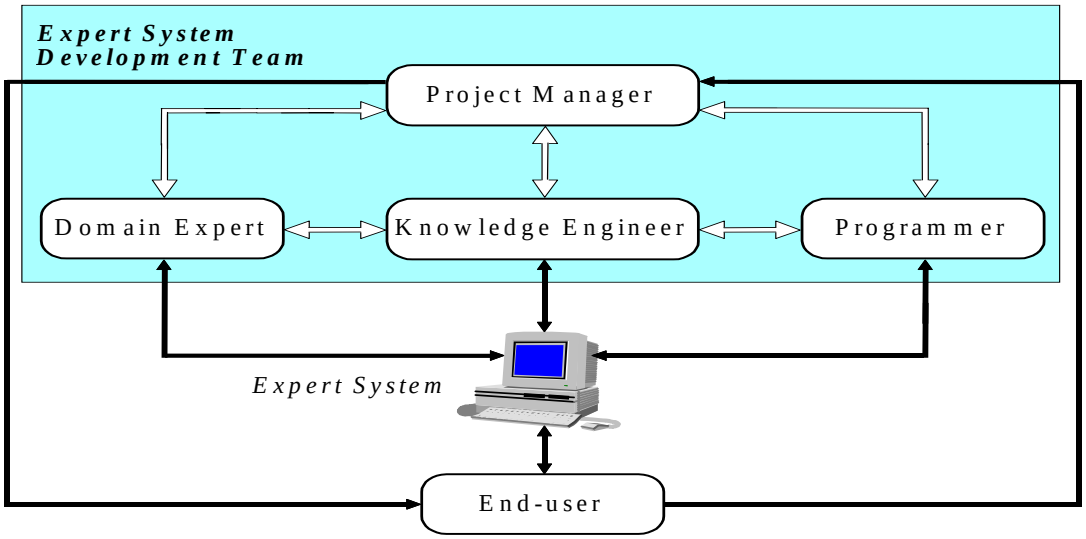

Figure 1: A real-world expert system team members (Negnevitsky, 2002)

Implementation

The adoption of an expert system development project in the Intelligent Systems subject at Victoria University was first undertaken in 2002, and the implementation reported here is of the 2003 and 2004 iterations of the project.

In a four week timeframe, students were asked to design and fully implement an expert system to play a game familiar to the students, known as 'Twenty Questions'. In a game of 'Twenty Questions' there are two players. One player thinks of an object, say a car, and the second player tries to discover what the object is, by asking the first player a series of questions. The second player is the winner, if they can guess the object, within twenty questions. If not, the first player is the winner. In the project, the expert system is the second player. According to Knowles, Holton, & Swanson (2001), students learn only if they are motivated and ready to learn and since there is a long association between the study of game playing and their solution strategies in AI and IS courses (Jones, 1987), it was expected that the creation of a game playing expert system would be a useful vehicle to arouse and motivate student interest (Pilgrim, 1995; Soh, 2004). In addition, students were encouraged to work in pairs rather than alone, so that they could discuss their ideas and strategies whilst they took on the differing roles of the various members of an expert system development team. Such role-playing gave the students an opportunity to also improve their oral and written communication skills (Sullivan, 1993).

The project was purposely broken into five sequential steps; the steps corresponded with the five roles in the expert system development team, domain expert, knowledge engineer, programmer, end user, and project manager. These steps gave students the opportunity to experience each role as shown in Figure 2. Each step came with instructions, and involved the use of differing technologies for completion. To further mimic real-world scenarios, the project instructions were quite general, and resembled directions rather than specifications. For instance, step three, titled 'Become a programmer', recommended students should acquire proficiency in an industry accepted expert system shell *Jess* (Sandia National Laboratories, 2003), but no specific instructions were given on how to use the software.

Steps

1. Become a **Domain Expert**.
   - Go to the 'Twenty Questions' web site http://q.20q.net/q.cgi
   - Play sufficient games to get a sense of the sort of questions which might be useful for deciding amongst objects.

2. Become a **Knowledge Engineer**
   - Go to the subject web page and download the file of 80 objects.
   - Design a set of rules, with the assistance of a domain expert that can be used to distinguish each object from the others.

3. Become a **Programmer**
   - Become proficient in *Jess*.
     - Complete the introductory *Jess* tutorial.
     - Investigate the documents directory that comes with the *Jess* software.
   - Investigate the examples directory that comes with the *Jess* software. There are possibly examples of how to set up your rule base and how to query your end user. Hint: animal.clp.
   - Code the rule base into *Jess*, load the database with the facts, and use *Jess* for your inference.

4. Become an **End User**
   - Thoroughly test the expert system, and try to break it.
   - Offer comment on poor design of questions or logic decisions.

5. Become a **Project Manager**
   - Get all of the above done before the deadline
   - Compile documentation

Figure 2: Expert system development specification

Aside from word processing and web page accessing, the main technology to which students gained exposure was *Jess*, a commercially licensed rule engine and scripting development environment for use in writing expert systems (Sandia National Laboratories, 2003). There were four main reasons for selecting the software: firstly, *Jess* supported different types of inferences, including types that were covered in lectures, i.e. forward chaining and backward chaining; secondly, *Jess* could be programmed to handle uncertainty in the rules which form the basis of any expert system; thirdly, *Jess* was written in Java, and the majority of the students in the IS subject were familiar with Java; and finally, *Jess* could be programmed to directly deal with, handle and manipulate other Java objects as part of the expert system graphical user interface(GUI).

Discussion

From a student perspective, given that an assignment was a mandatory part of the assessment, producing a game playing expert system was indeed a novelty. After playing some games of 'Twenty Questions' with the free software located at http://q.20q.net/q.cgi, (Burgener, 2003), some students were intimidated by the "intelligence" of the online game. The software asked a user to think of any object and it would then guess, with seemingly great accuracy, what this object was within twenty questions. Objects such as zebra, seaweed and train were easily guessed by the software and despite many attempts, students had great difficulty in finding an object that could not be guessed within twenty questions. Once students were reminded that their software had to decide amongst eighty known objects only, it became apparent that then their role as knowledge engineers was to decide upon the rules needed to discern one object from all of the others in the set.

After allowing time for students to complete, step 1, i.e., become domain experts, a classroom discussion was conducted on good design of the rule base. Using a binary tree search for the questions, the lecturer pointed out that theoretically it should be possible for all eighty items (in Appendix) to be categorized by a set of seven questions. This revelation stimulated much corridor discussion and querying by students, acting as knowledge engineers trying to decide upon the optimal set of questions to do the classification of objects for the game. Subsequently on a number of occasions, groups of students approached the lecturer asking for an adjudication about the legitimacy of some rules, such as *"Is it okay to ask if an object starts with the letter 'E'?"* and *"Can you ask a question that divides the remaining objects into three subgroups rather than two?"* Importantly, there was no single best or correct solution to the design since there are near infinite numbers of good strategies and rules that could be applied. This meant that, even with over fifty different student pairs, no two solutions should be identical and that any form of plagiarism would be obvious.

Having decided upon the set of rules for their rule base, the students next needed to act as programmers. In the lead time to the project, the students had been expected to complete a set of standard tutorial exercises to familiarize themselves with the basics of the *Jess*. These exercises were essentially a set of "how to" instructions on how to declare functions, rules and data. Frustratingly for the students, these exercises gave little indication on how to put the necessary sequences together to create their own game solution. Some students found the deliberate lack of detailed instructions particularly challenging. They had to search amongst the various help files and examples to learn how to combine these rudimentary steps into a coherent software solution. In the real world, ambiguous or scanty specifications, are one of the most likely of scenarios to be encountered by software engineers (Dawson, 2000). The programmer cannot expect to emulate a ready-made solution, or the development environment, or even know the language syntax already. The programmer will be expected to practice much initiative in acquiring these skills before production of any of the clients' code. Although many of the students found this step in the process the most frustrating, it was one of the most important, unwritten lessons that they were expected to learn.

Evaluation

Given the four week time frame for the delivery of the expert system, it was noted that as some of the weaker students began running short on time; it was the end user and project manager roles that tended to get postponed. Some students found that, as end users, they were often "too close" to the solution to thoroughly test their expert system and spot any design flaws. Additionally, these students spent little or no time producing adequate documentation for their product whilst acting as project managers responsible for completion.

For those students who did manage their time and project well, the payoff came in completion of a well constructed, working expert system along with the opportunity to extend their development. Such development was rewarded by a criterion based assessment system, as suggested by Lister and Leaney (2003), where a standard solution was considered worth only 70% of the total available marks. The remaining 30% of marks was given to encourage deeper learning by being assigned to any one of a possible range of extensions and enhancements to the solution. These extensions included exploration of *Jess's* capabilities of handling uncertainty as well as using Java to produce more "fun" graphical user interfaces for the system's front end. Several more motivated students did incorporate these extensions into their project, improving the functionality of their expert system and receiving additional marks dependent upon the degree of difficulty and the innovation involved. An example of a student expert system is shown in Figure 3.

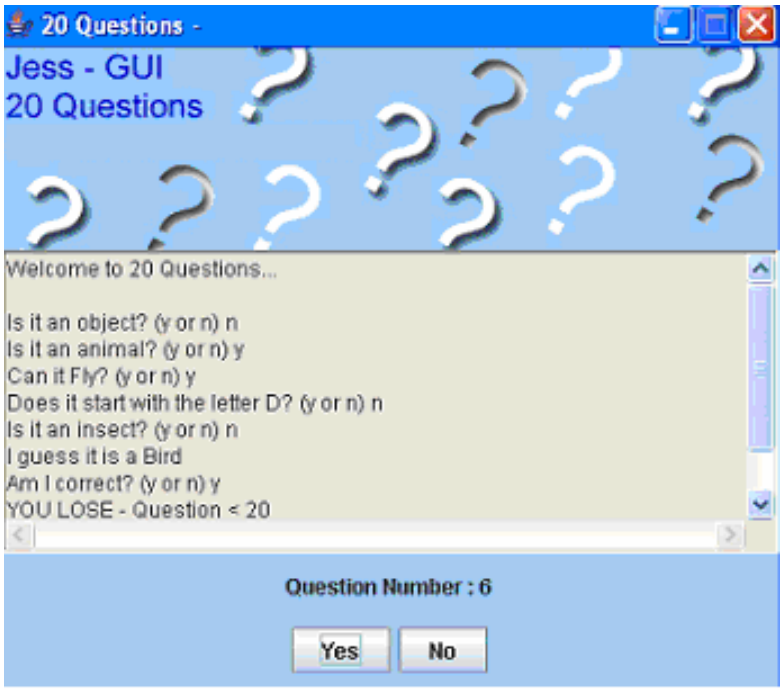

Figure 3: A student expert system team with a GUI front end

The pursuit of adventurous teaching and active learning has been advocated by McLoughlin and Oliver (1999), but was the inclusion of a knowledge engineering project worthwhile in this instance? Standard student end of subject surveys included two statements in 2003 and three statements in 2004, referring to the assignment, see Table 1. From a student perspective, it seemed that the assignment was useful in giving an appreciation of the process of creating a rule-based expert system and in particular seemed to achieve an appreciation by students of each of the roles they had been asked to play. The student responses relating to the easiness or otherwise of the task difficulty were well spread. This result was partly to be expected, given the diverse backgrounds of the student body. In future iterations, we plan to survey the students more extensively about the project, with both pre and post assignment surveys to gauge the actual learning outcomes.

Table 1: Intelligent Systems Student Survey results for 2003 and 2004. Total of 58 responses in 2003 and 46 responses in 2004

Questions	Disagree/ Strongly Disagree		Agree/ Strongly Agree	
	2003	2004	2003	2004
The assignment was easier than I expected	20	19	15	8
The assignment taught me about all aspects of creating a rule-based expert system	7	6	43	30
The assignment taught me the different roles of a knowledge engineer, domain expert, programmer, end user and project manager in creating an expert system.	–	4	–	30

The Intelligent Systems subject and its parent Computer Science degree that has been described above runs both locally in Melbourne, and 'off-shore' in Hong Kong, China, Malaysia and in Sydney, Australia. Since the incorporation of the knowledge engineering project in the local course, the same assignment has been set for all offshore groups in 2004. Upon the advice of their 'local' lecturers, the roles remained the same. However, it was necessary to change a rose to an orchid, this flower being more familiar to our offshore students. From all reports to date, the project seems to have traveled well and a more extensive investigation across campuses is planned for the near future.

Conclusions

Knowledge engineering, the development of an intelligent system, is an art, as much as it is a science. In an undergraduate IS subject, what is the best way to prepare students in this art, i.e., learn something of the process and involve them in product creation? Supported by standard lectures and laboratory sessions, the inclusion of an expert system development project was an ideal real-world scenario that encouraged students' learning and practice. In 'becoming a star', i.e., role-playing each of the five members of an expert system development team, students were able to apply much of the theory of problem solving strategies, whilst experiencing the tasks of, and the demands upon, each team member. Also, the effectiveness of this project was due, in part, to the fun aspect of producing a game that motivated students' curiosity and continuing engagement. As a result, the 'become a star' project has become pivotal in future iterations of the IS subject, both locally and offshore.

Acknowledgements

In particular, thank you to the overseas 'local' lecturers of the Intelligent Systems subject, who supported their/our students in their expert systems developments.

References

Armarego, J. & Clarke, S. (2003). Preparing students for the future: Learning creative software development - setting the stage. In *Learning for an Unknown Future: Proceedings of HERDSA* (The Higher Education Research and Development Society of Australasia) Conference Christchurch, New Zealand.

Burgener, R. (2003). 20Q Twenty questions. V5.3.6. Retrieved Jan 12, 2004 from http://q.20q.net/q.cgi

Cawsey, A. (1994). MYCIN: A quick case study. Retrieved Jan 12, 2004 from http://www.cee.hw.ac.uk/~alison/ai3notes/section2_5_5.html

Dawson, R. (2000). Twenty dirty tricks to train software engineers. In *Proceedings 22nd International Conference Software Engineering.ACM* (pp 209-218). Limerick, Ireland.

Dean, N. & Hinchey, M. G. (1995). Introducing formal methods through role-playing. In *Proceedings of the twenty-sixth SIGCSE technical symposium on Computer science education.* 3/95 (pp.302-305). Nashville, Tennessee, U.S.A.

Hill, J. M. D. & Alford, K. L. (2004). A distributed task environment for teaching artificial intelligence with agents. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education*. Norfolk, Virginia, U.S.A.

Jackson, P. (1999). *Introduction to expert systems* (3rd ed.). Addison-Wesley.

Jones, J. S. (1987). Participatory teaching methods in computer science. In *Proceedings of the eighteenth SIGCSE technical symposium on Computer science education*. (pp. 155-160). St. Louis, Missouri, U.S.A.

Knowles, M., Holton, E. & Swanson, R. (2001). *The adult learner: The definitive classic in adult education and human resource development* (5th ed.). Woburn, MA: Butterworth-Heinemann.

Kolb, D. A. (1984). *Experiential learning: Experience as the source of learning and development*. Englewood Cliffs, NJ: Prentice Hall.

Kolb, D. A. & Fry, R. (1975). Towards an applied theory of experiential learning. In C. L. Cooper (Ed.), *Theories of group process*. Chichester: John Wiley.

Lister, R. & Leaney, J. (2003). First year programming: Let all the flowers bloom. In *Proceedings of Fifth Australasian Computer Science Education Conference, (ACE2003),* Adelaide, Australia. 20/03.

McLoughlin, C. & Oliver, R. (1999). Pedagogic roles and dynamics in telematics environments. In M. Selinger & J. Pearson (Eds.), *Telematics in education: Trends and issues* (pp. 32-50). Oxford: Pergamon.

Negnevitsky, M. (2002). *Artificial intelligence: A guide to intelligent systems* (1st ed.). Pearson Education.

Pilgrim, R. A. (1995). TIC-TAC-TOE: Introducing expert systems to middle school students. *SIGCSE Bulletin, 27* (1), 340 – 344.

Sandia National Laboratories. (2003). Jess the rule engine for the JavaTM platform. Retrieved Jan 12, 2004 from http://herzberg.ca.sandia.gov/jess/index.shtml

Soh, L. K. (2004). Using game days to teach a multiagent system class. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education*. Norfolk, Virginia, U.S.A.

Sullivan, S. L. (1993). A software project management course role-play team-project approach emphasizing written and oral communication skills. In *Proceedings of the twenty-fourth SIGCSE Technical Symposium on Computer Science Education*. Indianapolis, Indiana, U.S.A. 2/93 (pp. 283-287)

Ye, L. R. & Gray, G. L. (1996). Developing self-monitoring intelligent tutoring systems for accounting education: Some preliminary results. In *Proceedings of the Australasian Society for Computers in Learning in Tertiary Education 1996 Conference*, Adelaide, South Australia.

Zowghi, D. & Paryani, S. (2003) Teaching requirements engineering through role playing: Lessons learnt. In *Proceedings of the eleventh IEEE International Requirements Engineering Conference*. Monterey Bay, California, U.S.A.

Appendix

List of 80 objects to be classified

apple	bird	cake	coat
arm	boat	car	corn
ball	book	cat	cow
banana	box	chair	cup
bed	branch	chicken	dinghy
bell	bread	clock	dog

doll	hill	paper	shoulder
dolphin	hoof	pencil	sofa
door	horse	plant	squirrel
duck	kangaroo	postcard	stick
egg	leg	potato	table
elbow	letter	rabbit	tent
elephant	lily	rat	thermometer
eye	milk	salt	toy
farm	money	school	tree
fish	mosquito	seed	water
foot	mountain	sheep	wombat
garden	orange	shelf	wood
grass	orchid	ship	wool
hand	panda	shoe	zebra

Biography

Anne Venables is a lecturer in Computer Science at Victoria University. She has research and teaching interests in artificial intelligence and intelligence systems. As a former secondary Science and Mathematics teacher who has migrated into tertiary education, Anne is also interested in innovations in education and has previously published in this field.

Grace Tan is a lecturer in Computer Science at Victoria University. Her research interests include innovative teaching methods, development of graduate attributes, and issues related to female students in computing courses and has published in these areas.