Teaching to Foster Implicit Knowledge

Errol Thompson[1]
Information Systems, Massey University
Wellington, New Zealand

Abstract

Michael Polanyi says, "we know more than we can tell". Polanyi is arguing that there are many things that we do based on tacit or implicit knowledge. Boisot describes a social learning cycle where implicit knowledge used in the work place is codified or made explicit so that it can be passed on to others. Boisot argues that the new learner then absorbs this and gradually internalized through testing and use in a wide variety of contexts. This becomes part of their way of working and generates new 'tacit' knowledge. In software development, the structure of the software is often dictated by the developer's experience. If a new development environment is encountered then the developer will attempt to apply past strategies in the new environment. This paper contends that in teaching software development skills, we are endeavouring to foster the development of a 'tacit' knowledge base that the learner can then apply in future projects. The role of education becomes one of changing the learner, which is achieved through changing their assumptions or 'tacit' knowledge base. This paper reviews the literature to explore the 'tacit' or implicit knowledge as it applies to information systems topics and discusses initial investigations of how it applies to the teaching of programming. The particular focus is on the need for the students to develop an implicit understanding of the topics so that it becomes part of their way of thinking about the subject and becomes part of their work pattern.

Keywords: tacit knowledge, implicit knowledge, learning

1. THEORETICAL BACKGROUND

When Michael Polanyi (1966) argues for tacit knowledge, he is contending that there are many things that we do that we would struggle to fully explain to others. We act on a form of knowledge that is derived from experience and not necessarily from explicit learning or codified knowledge. Polanyi argues for the involvement of the person in the act of knowing and that our knowledge foundation is based on a rational commitment to what we perceive as being known. He says, "I regard knowing as an active comprehension of things known" (Polanyi 1958: vii).

In the field of knowledge management, it is recognized that companies operate on unwritten rules and the tacit knowledge of their employees (Hall 1997). Some of this knowledge is critical to the competitive advantage of the organization. To avoid loss, organizations seek to codify this knowledge to make it more readily available to other employees. Codification or making explicit does not ensure that the knowledge is then passed on, or becomes part of the way other employees operate. There needs to be a learning cycle that ensures that the codified knowledge is transferred and that those who receive it begin to apply this knowledge. However, every employee comes to a task with an existing knowledge base (their tacit knowledge) of that task. When new knowledge is imparted, they combine this knowledge to generate new tacit knowledge that will alter the way that they approach the task and it will possibly be different to the original employee's tacit knowledge. The cycle of codification and knowledge transfer begins again. This cycle is known as the social learning cycle (Boisot 1995).

[1] E.L.Thompson@massey.ac.nz

Schön (1983) in arguing for reflection-in-action contends that the practitioner reframes a problem situation in an attempt to utilize prior experience of a familiar situation. To achieve this, the practitioner utilizes their tacit or implicit knowledge of their field of operation. In the process, the practitioner is developing their implicit knowledge base to be able to handle new problem situations.

2. WHAT IS "TACIT" KNOWLEDGE?

Tacit knowledge is that knowledge which a person uses to accomplish tasks but has not brought into conscious focus. When challenged, the practitioner may have difficulty expressing the knowledge. The practitioner may even be unaware of the knowledge that they have utilized and the source of that knowledge.

An individual's implicit knowledge base will direct their initial approach to a task and their ability to comprehend new situations. What is expressed as explicit knowledge today may be implicit knowledge in tomorrow's activities.

Some examples

A cyclist balances by riding in small arcs. In initial learning, these are obvious wobbles. As riding experience increases the tiny movements to balance the bicycle are no longer noticeable yet they still happen. The cyclist's focus shifts to other technical aspects of their riding and the need to balance becomes an implicit pert of their riding behavior. For the competitive cyclist this may see a change to a focus on cycling technique as especially how to maintain a higher speed with lower energy outputs.

The cyclist implements a training program based on assumptions about how to achieve the desired competition results. If the initial training program is based on the assumption, that higher gear ratios increase speed and later the cyclist discovers that higher cadence in lower gear ratios enables faster acceleration and greater endurance then the cyclist must retrain both his/her body and mind to the new strategy. A failure, to train in the intended racing strategy, leads to using the training strategy when the pressure is on during racing. Even though the cyclist may have understood explicitly the new strategy, the implicit strategy, developed during training and that has been used in past racing, will dominate when the race pressure increases.

The novice programmer struggles with the basic logic constructs and the structuring techniques for their program code. The syntactical structure is a constant struggle. When they read existing program code, their focus is on the detail and not on the overview.

In contrast, experts are able to recognize patterns that exist in their field of expertise to apply their experience to the situations and problems that confront them (Chi et al 1988). This often makes the expert appear as though they are doing minimal analysis of the situation before making a decision or taking action. Experienced programmers rapidly understand program code without detailed analysis and without prior knowledge of the programming language. Solway et al. (1988) contend that expert programmers use "programming plans" or schemas to comprehend program code.

While the novice analyst struggles with identifying entities and the relationships between entities, the expert analyst recognizes many entities in the dialogue of the customer. The expert also recognizes the patterns to be applied to the solving of a particular customer problem. The implicit knowledge base of the expert seems to allow the expert to jump stages in the analysis process. This can confuse the novice because they do not understand how the expert came to their conclusion.

In test driven development, the expert developer has an implicit understanding of the types of tests that should be used to drive development. Novices, as evidenced through questions and discussions in agile development mailing lists, often write tests that cover large chunks of code. The novice as a consequence may recognize the explicit advantages of a test driven approach but struggle to realize those advantages in practice because of the difficulty in identifying appropriate tests to use. Here the required implicit knowledge is the appropriate thinking patterns to enable the problem space to be divided into appropriately sized testable chunks.

Schön's reflective practitioner (1983) will utilize those patterns of knowledge that they have built up from experience. They utilize these patterns to reframe problems in order to arrive at a solution. To change the applied knowledge, the reflective practitioner must do more than read texts in the field. They must build a new tacit knowledge base that incorporates new approaches and ideas through application to new environments.

Summary
A person's tacit knowledge base expands and the focus of their attention changes. The tacit knowledge base provides the base know-how and thinking patterns for completing a task. It provides the applicable patterns to apply. Expanding the implicit knowledge base is fundamental to the development of expertise.

3. SOFTWARE DEVELOPMENT

In approaching software development tasks, experienced developers bring with them a wealth of knowledge. Some of this knowledge may be expressed explicitly. I would contend that a large portion of it has become tacit knowledge. A significant element of this tacit knowledge is how the expert thinks about programs and the programming task. Like the cyclist, it is the tacit knowledge of software development that the expert relies on in pressure situations.

Program code is written to known logic patterns and data structures. These patterns form part of the implicit knowledge toolkit of the experienced programmer. Where an unfamiliar process is to be implemented, the programmer will look for examples or use code generation tools to develop a solution. The experienced software developer will dissect the problem into smaller chunks and reframe it using known program patterns.

To bring change, educators need to assist students to build a tacit knowledge base of how programs are coded, or change the logic patterns for writing program code. When teaching a first programming language, the educator can focus on the syntax and techniques of the language. Some students will develop programming skills from this base but others will struggle to comprehend the nature of the task. They struggle because they may have no understanding of the concept of a program, or of the semantics of the language. Others may struggle because they have no understanding of the logic patterns and data structures for the assigned programming tasks.

The experienced programmer, when approaching a new language, takes with them the coding and data structure patterns or programming plans that have become implicit in their approach to programming. However, the experienced programmer will still struggle with a new language or coding environment where the coding paradigm or supporting code libraries do not conform to their implicit understanding of how programs are structured (i.e. program plans). This is reflected in the struggle that some programmers have in moving from structured programming environments to object-oriented or object-based environments.

An example
In a course teaching Pascal programming to undergraduate students, 50% of the students were either withdrawing from the course or failing to pass the assessments. Those involved in the course assumed that programming is difficult to teach.

The course utilized progressive programming exercises that went from simple programs (a sequence of input, process, and then output) and built to complex programs. The students were introduced through handouts and brief lectures to each of the required language constructs. The students seemed to miss the connection between the use of the constructs and how to build a program that used them.

The next step in the course development was to run theory sessions. In these theory sessions, the language constructs were introduced, the logic patterns (Dale and Weems 1992; Thompson 1992), and selected examples from the progressive programming exercises worked through. When working through the examples, emphasis was placed on the reasoning of the logic patterns. The lecturer endeavored to make explicit the thinking behind the construction of the programs. When a loop was used, the lecturer would ask, "What needs to be done to initialize the loop control?" or "What needs to get updated within the loop?"

The results of this teaching process were an increase (to an 80% pass rate) in the number of students completing and passing the course. The only difference in the assessment process was that the students had to provide proof to the lecturer that they had completed a set number of the progressive programming exercises along with a major assignment and theory test. The assignment was of the same level of difficulty as the original assessments and moderated by a peer lecturer.

The final part of the assessment was a theory test in which the students answered questions that focused on the theory of programming. This enabled the lecturer to test how well students understood the logic patterns and programming concepts.

Explanation for results

Why should the change in teaching approach deliver a dramatic change in results? The process did involve the lecturer making explicit the thinking processes and logic patterns that were behind the coding approach. In terms of the social learning cycle (Boisot 1995), this is the codifying and making explicit steps. The student then completed a series of progressively harder programming exercises that used the thinking processes and logic patterns and forced them to complete repeated practice of the concepts. These exercises helped the students develop their implicit understanding of concepts and application to programming.

Within the teaching approach, there is also an element of constructivism. The students as they work on the programming exercises are revisiting the theory presented by the lecturer. As a result, they actively construct their own knowledge base for the task. This leads to the student developing their own understanding and way of looking at and thinking about the programming task (Biggs 1993).

4. LEARNING A NEW PROGRAMMING LANGUAGE

In the last two years, I have been requested to teach programming courses for second and third year classes using programming languages that I have had minimal knowledge of. Being an experienced software developer and having learnt a wide range of languages, the task did not appear to be difficult. It was reasonably easy to learn the core language constructs but I stumbled in endeavoring to teach the languages through lack of experience with the language, lack of awareness of the rich object libraries or frameworks that are now available, and a lack of awareness of the logic patterns for object-oriented and event driven code.

My foundation in logic patterns enabled me to learn the constructs for assignment, conditionals, loops, and procedure calls. Even the development of simple event driven screen forms proved relatively easy. I couldn't communicate how I thought about programming in this new environment. I needed a better base knowledge built from completing a number of practical programming exercises and from understanding the design patterns used in developing event driven code and in the frameworks.

During the teaching of the courses, my knowledge increased as I endeavored to help students and to develop a set of progressive programming exercises. In explanation to students, I was forced to think about how I could explain the logic patterns of object-oriented and event driven programming. My own strategies for learning programming languages were being rapidly revised. Past experience and thinking patterns in procedural and structured programming languages had not prepared me for the new system architecture structures of these object-based and event-driven software development environments and frameworks.

I needed to be introduced to the design patterns and to make the thinking behind the design patterns part of my thinking and approach to teaching. The procedural and structured programming patterns and thinking paradigms are inadequate but not irrelevant for the new environments. As I have developed my own thinking of program and application architectures for these primarily object-oriented and event driven environments, it has become easier to communicate the thought processes or implicit knowledge necessary to develop the required programming skills.

5. MODELING

Armour (2000a) describes programming as the capturing of business knowledge and encoding it in software. The software developer acquires knowledge (Armour 2000b) and models that knowledge in terms of data structures, processing logic, processing patterns, and the software architecture. Like Schön's reflective practitioner (1983), the software developer has to draw on their experience of software patterns to develop a model or representation that accurately captures the business knowledge.

Students initially have difficulty comprehending how to encode business knowledge because often they are struggling to comprehend the knowledge requirements of two domains. They lack the experience or tacit knowledge base in the software development tools and techniques to be able to easily apply these to the new knowledge domain of the business environment.

As well as lacking a tacit knowledge base of the data structures, processing patterns, and software architectures, the learner also lacks a thinking framework for software development and systems. Without the thinking framework, they are unable to reframe the business knowledge that they are acquiring into the programming patterns or system patterns that can be used to represent the knowledge.

Pirsig (1974) describes a similar problem with students not knowing what to write on a topic other than reflecting back what they remember they have been told or heard. Pirsig argues that they needed to be encouraged to trust their own perception and existing knowledge. They had to be encouraged to write about what they know from their own experience. The ingredient missing is the thinking patterns or cognitive skills to translate a concept into another form or to connect concepts with prior knowledge.

A first step in many of these situations is to help students recognize what they already know (e.g. their existing understanding of what a program is outside the software setting). That is the lecturer has to help them make explicit their tacit knowledge (Collins 1979). The students have to be encouraged to reconstruct their understandings and to be confronted with the thinking processes that enable transformation of concepts.

6. PEDAGOGICAL IMPLICATIONS

Collins et al. (1989) describe the use of cognitive apprenticeship. In cognitive apprenticeship, the learner or apprentice is given the opportunity to observe the process before being assigned the task. After having observed the process, the learner attempts to complete a series of tasks with a decreasing amount of support or with the tasks growing in increasing complexity but utilizing the same base concepts or cognitive skills and tools (Rosson and Carroll 1996).

The cognitive apprenticeship step that is often missed is the opportunity for the learner to observe the master at work. A repeated exercise or prepared solution exercise often misses the spontaneity of thinking of the master practitioner and therefore lacks the realism that enables communication of the cognitive processes involved. A key part of the observation is seeing the reasoning that brought about the solution.

Also the support given to the learner in their early attempts at the task must foster the development of the learner's cognitive skills. The solution to the task doesn't help the student to understand the reasoning that made that solution valid. It is the reasoning or cognitive process that is more important for the learner's ongoing ability to be able to complete the task.

7. LECTURER ROLE IMPLICATIONS

In this type of environment, the lecturer increasingly moves away from being the source of the required knowledge to being the facilitator of learning and to being the master practitioner. In this role, the lecturer needs to have an implicit understanding of the subject matter and the related cognitive skills. From this knowledge base, they are able to quickly evaluate a learner's work, reframe it into a programming thinking pattern, and raise questions that will encourage the learner to explore their own understanding and the subject further.

The lecturer needs to be analyzing or reflecting on how programmers think about programming and communicating those thinking

processes. This meta-cognitive approach enables the lecturer to stimulate the cognitive skills of the learner.

What level of knowledge is required by the lecturer and tutorial staff in order to foster the development of the higher cognitive skills? Does a lack of in depth knowledge cause the lecturer to utilize advice offering teaching styles that lead to standard solutions? The lack of familiarity may also limit the lecturer's ability to model the cognitive skills that the learner needs to acquire. Further research on these aspects is required.

If the lecturer is unable to model the cognitive skills or stimulate the learner to develop the required cognitive skills then the learner might be able to complete a specific programming task but may fail to be able to translate that knowledge to a new situation or to be able to select an appropriate solution from a range of possible solution options or to explain their program code. Without developing the implicit knowledge base and related cognitive skills, the learner will continue to struggle with the subject matter and to be unable to apply the learning in practice.

8. CONCLUSION

How a programmer thinks about programming is part of their implicit knowledge base. It is the base from which they can draw solutions and through which they can translate programming problems to develop an understanding of possible solutions. Programming apprentices or learners have to develop these cognitive skills and build them into their implicit knowledge base.

The lecturer must demonstrate and make explicit these cognitive skills as a first step in developing the learner's cognitive skills. Further, the lecturer must challenge the learner to think through to solutions utilizing the thinking patterns and not simply to look for example solutions.

9. REFERENCES

Armour, Phillip G. (2000a) "The case for a new business model: Is software a product or a medium?" Communications of the ACM 43 (8), pp. 19-22.

Armour, Phillip G. (2000b) "The five orders of ignorance: Viewing software development as knowledge acquisition and ignorance reduction." Communications of the ACM 43 (10), pp. 17-20.

Biggs, John B. (1993) "From theory to practice: a cognitive systems approach." Higher education research and development 12 (1), pp. 73-85.

Boisot, Max H. (1995) "Is your firm a creative destroyer? Competitive learning and knowledge flows in the technology strategies of firms." Research Policy 24, pp. 489-506.

Chi, Michelene T.H., Robert Glaser and M.J. Farr (1988) The nature of expertise, Hillsdale, NJ: Lawrence Erlbaum Associates.

Collins, Allan. (1979) Explicating the tacit knowledge in teaching and learning. Cambridge, MA: Bolt, Beranek and Newman, Inc.

Collins, Allan, John S. Brown, and Susan E. Newman (1989) "Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics." In: Resnick, Lauren B., (Ed.) Knowing, learning, and instruction: Essays in honor of Robert Glaser, Hillside, New Jersey: Lawrence Erlbaum Associates

Dale, Nell and Chip Weems (1992) Introduction to Pascal and structured design, 3rd edn Turbo Pascal version. Lexington, Mass: Heath.

Hall, Richard (1997) "Complex systems, complex learning, and competence building." In: Sanchez, Ron and Aimé Heene (Eds.) Strategic learning and knowledge management, pp. 39-64. Chichester: John Wiley & Sons Ltd

Pirsig, Robert M. (1974) Zen and the art of motorcycle maintenance, London: Corgi Books.

Polanyi, Michael (1958) Personal knowledge: towards a post-critical philosophy, Chicago: Routledge and Kegan Paul Ltd.

Polanyi, Michael (1966) The tacit dimension, Gloucester, MA: Double Day and Company.

Rosson, Mary B. and John M Carroll (1996) "Scaffolded examples for learning object-oriented design." Communications of the ACM 39 (4) pp. 46-47.

Schön, Donald A. (1983) The reflective practitioner: how professionals think in action, New York: Basic Books.

Soloway, Elliot, Beth Adelson and Kate Ehrlich (1988) "Knowledge and processes in the comprehension of computer programs." In: Chi, Michelene T.H., Robert Glaser and M.J. Farr, (Eds.) The nature of expertise, pp. 129-152. Hillsdale, NJ: Lawrence Erlbaum Associates

Thompson, Errol (1992) CBC-PP100 programming principles workbook. Auckland, Carrington Polytechnic.