

# The Karate Kid Method of Problem Based Learning

Fred Fonseca and Larry Spence

**Abstract** The traditional introduction to computer programming course delivers lectures that meticulously describe language components and the way they work combined with homework and laboratory drills. The challenge of teaching introductory programming courses is: the students do not know enough to work on interesting and challenging projects. As a result, they are assigned small structured (toy) problems that do not engage their curiosity and allow them to try on their own. This results in little motivation and poor learning. Large numbers of students fail, drop, or complete courses without learning to program. In contrast, this introductory class enables students to learn programming languages by designing and implementing a computer game. Two obstacles must be overcome to make this work. The first is the students' anxiety when faced with a complex task they cannot do. The second issue is the menial and tedious practices they must undergo to master the tools required by the task. Imbedding structured problems within the requirements of a complex unstructured project helps resolve these issues. We describe this version of PBL using the pedagogical metaphor of the popular movie, the Karate Kid.

## Introduction

It is the fifth week in a Java programming course. Some students plunge into the project. Others wait in suspended animation. Instead of programming, some students surf the net or catch up on e-mail. The instructor approached one such student and asked:

Why aren't you working on the project?

I am waiting for you to teach us how to do it.

But you already know everything you need to know.

No, I don't know anything. What I know is that you will teach us the most important things only by the end of the semester.

So please tell me what you want to do in your project that you don't know how to do?

---

F. Fonseca (✉) · L. Spence

College of Information Sciences and Technology, The Pennsylvania State University, 330 D IST Building, University Park, PA 16802, USA

e-mail: fredfonseca@ist.psu.edu

The student answered that he wanted to create an introductory screen and then make it vanish replaced by a screen that offered alternative game settings. The instructor replied that problem #6 on layouts covered the operation. The next challenge was how to transfer the settings from the options screen to the real game screen. Again the instructor reminded the students that a set of three exercises: homework #4, homework #5, and problem #7 focused on exchanging information between classes in three different ways. This is the transition pattern: student questions, assignment reminders, followed by application to the real problems of software production.

The Karate Kid movie dramatizes this transition problem. Bored with practicing basic movement Daniel (the “Karate Kid”), charges that Miyagi hasn’t taught and he hasn’t learned. The master responds: “Show me sand the floor. Sand the floor. Sand the floor. Big circle. Now show me wax on, wax off. Wax on, wax off. Wax on, wax off! Wax on. ...wax off.” As he shouts Miyagi strikes from above, and then kicks from below. As the kid tries to avoid blows, the master explains, “You see, this is paint fence”. “This is sand floor.” “This is wax on, wax off.” When Daniel connects the motions of practice with the blows that ward off the master’s attack, he realizes that he knows karate.

In the movie, practice is initiated with ordinary tasks to prepare a novice to learn basic skills in anticipation of martial combat. That method—imbedding practice on basics in the context of a professional project—guided our implementation of an introductory programming course. We adopted an extensive re-design of an existing programming course requiring more student initiative. We reasoned that just as we don’t expect anyone to learn French by listening to lectures on syntax and vocabulary, we can’t expect students to learn programming by watching demonstrations and memorizing definitions. Thus wholesale Problem Based Learning seemed the best approach.

Instead of rote memorizing of language syntax which is a common approach in the teaching of computer languages, our approach focused on the teaching and learning of basic concepts. For instance, in previous versions of the course there was a special chapter on *procedures, functions, and methods*. These are basic concepts in object-oriented programming. This part of the course described all the concepts and explained the definitions of each one. There were small examples, with localized pieces of Java code to show the use of each concept. But this was not enough. The students could still memorize and do well in the quizzes while still not really understanding them.

What happens in the revised PBL version of the course is that the students learn the same concepts through using them. Through practice with mini-problems students become familiar with the basics. In attempting to complete a project students get stuck in places where they need to solve the same kind of problems faced by the creators of the language. They learn solutions to the recurring problems of software engineering. The students are in a controlled environment in which they have the support and structure to help them. The students learn the same concepts that were taught before and while they may not be able to define or name them initially they understand how and why they are used. The instructor’s job is to formulate good problems that let the concepts emerge with practice. Ultimately, the best problems

are similar to the ones that programmers faced in generating the language's solutions. As students practice the same questions appear over and over again in different sections of the course.

## Problem-Based Learning

Problem-based learning (PBL) is a new edition of an old idea: learning should take place in a realistic context in which students get their minds on a problem (Dewey 1938; Kilpatrick 1918). In traditional classrooms learning goes on for the motivated mostly in the dorm, the library, or the laboratory. In contrast, PBL centers learning in the classroom where students can practice with an instructor to guide them. Despite the good sense and promise of PBL, implementation is difficult. Major problems revolve around three issues: (1) the altered roles of instructors and students, (2) the cultural inertia encountered in classroom designs and course schedules, and (3) the quality and characteristics of problems.

While exploiting the insights of practitioners and scholars, we chose to focus on problem structure as a way to address the issues. We found two common mistakes. Some instructors import the real world into the classroom with messy problems and students fail and flounder. Other instructors provide more structured lab problems and group discussions between lectures. In an introductory programming class, this seems to encourage student to memorize definitions and algorithms without learning programming skills.

The cognitive demands of PBL also shock students. Accustomed to solving problems by applying what they have learned, students see tackling problems in order to learn as "backward teaching." It challenges everything they think they know about education. "How", they ask, "can we attempt problems before professors have told us what to do?" Since it makes them seem incompetent and demands great initial effort, students resist. They translate the PBL acronym to mean pain-based learning.

Grafting PBL problems onto traditional courses incrementally seems a sensible strategy to dampen the shock. It isn't usually successful. One author, as director of a learning innovation centre, worked with more than 100 funded faculty members who introduced new techniques, including PBL, piecemeal into their courses. Discouraged by the paucity of results and a lack of institutional support, instructors gave up. Few innovations lasted more than three iterations.

Dean Ornish, a heart specialist, who tries to induce changes in patients' lives to keep them alive, has found that when people try to make a few modest changes they get the worst of two worlds. They lose the satisfaction of old habits but gain little reward from small changes (Ornish et al. 1990). Similarly, we think that attempts to smooth the transition to PBL by adding problems to traditional designs produce meagre results, frustrating students and instructors alike. We opted for a full PBL implementation right at the beginning of our course revision.

The course we are describing in this chapter is called *Introduction to Computer Languages*. It is offered in multiple sections every semester for approximately

150 sophomores. The average class size for the sections discussed in this chapter is 41, with a range of 21–64 students. A major pre-requisite is an introductory Computer Science course usually covering C++. The course introduces different computer languages; including Java script as a programming language, UML (Unified Modeling Language) as a language used for the design of applications; and XML (Extensible Markup Language) as a language used for data exchange.

## The Karate Kid Design

We wanted to develop a learning experience that presented students with a project realistic enough to excite interest, while providing structured problems for them to practice and learn the basic skills. We found a good metaphor to approach these issues in the original *The Karate Kid* movie. In that film an eager teenager, Daniel Larusso, wants to defend himself against Karate bullies. An aging Karate master, Keusuke Miyagi, makes a deal with the bullies' teacher to lay off Daniel in return for a promise to face them in tournament combat weeks away. Daniel demands to start by punching. Miyagi, instead, assigns him to wax and polish antique automobiles, sand a deck, and paint a fence. After several days, he takes the student to the beach to struggle for balance in surf. Daniel repeatedly asks, "When do I learn to punch?"

Miyagi designs the exercises to build on the student's prior skills to introduce basic Karate movements. While the student polishes, sands, paints, and plays in the surf, Miyagi gives novel instructions such as: *Polish clockwise with the right hand and counter-clockwise with the left. Paint up and down with the right hand and back and forth with the right. In the surf, keep upright no matter what happens.* Daniel unknowingly learns to deflect blows, strike, and maintain balance. None of this looks like martial arts to him.

The novice's first impulse is to do something. The likely results are mistakes. Deny the impulse and you throttle the desire to learn. Allow a cascade of failures and you destroy confidence in being able to learn. Instructors then have to build on the desire to learn; not dampen students' curiosity or abandon them to flounder (Carroll and Rosson 1987). That requires instructors to use something like the *Karate Kid* model.

The model is: (1) provide examples of and promote a desire to master new knowledge; (2) create exercises that involve what students already know, but demand changes; (3) help students relate what they learn about basics to a realistic project; and (4) give them opportunities to try out new skills by performing applications. In short, the model embeds basic practices within an expert performance.

On the first day of class, the instructor introduces the anchoring idea of the class—a semester-long project to design and program a computer game. Students examine and evaluate examples of applications created in past classes. This shows them what they need to be able to do and what they can do even as it challenges them to think beyond their current abilities. In teams of three, they begin to conceive of

a computer game. The game functions as a course map. To complete the big project they must master the minutiae. Students work back and forth from the map to the details and from the details to the final project itself.

For example, students can develop a search game in which the player tries to find a stealth professor on campus. Or they can depict a classroom in which the goal is to keep students awake and alert under varying conditions. Other examples are: a flying saucer piloted by different professors each with different skills trying to rescue students; a student trying to get to a football game by collecting different objects and navigating obstacles; and, of course, variations on maze running. All games involve familiar elements of student life such as campus locations, undergraduate pastimes, professor's behaviours, video games, athletic events, or planning parties.

The gap between the requirements of programming the game and current skill levels creates the most trouble. To avoid failure, students drag their feet (Dweck 2006, p. 52). To overcome this they have to believe that work will get them to the necessary skill levels. The instructor must convince them that working hard will make them smarter without promising that it will be easy.

To lessen anxiety, the instructor attempts to design assignments from the student's point of view. The variety of ways that students attack problems, the varied backgrounds and skill levels in the class, and the different speeds of learning require problems with three levels of difficulty. Students with programming experience can move quickly to the highest level. Beginners may need more time at the lowest. Ample time to solve each problem is crucial. All this requires monitoring and managing of activities. Often students think they are behind while, in fact, they have taken a different learning path (Fischer and Rose 2001). The instructor strives to make clear that the goal is for each student to improve. Students need not progress in lock-step. This classroom culture offsets challenge with ease of first steps.

## ***Phase 1: Scaffolding***

The instructor's role changes with each phase of the course—the scaffolding phase is instructor driven; students drive the performance phase.

We use the term scaffolding to describe instructional practices that encourage students to play and practice without fear of errors. This is Karate Kid scaffolding—start with tasks that students can do, but interject novel demands to push them to the next level of understanding. This design dampens the consequences of early mistakes but doesn't eliminate the possibilities of trial and error.

For example, Daniel's recognizable errors when polishing, sanding and painting are minor—missed places on a car's hood, leaving sanding marks against the grain, paint drips or a mouthful of salt water. Miyagi doesn't ask the student to attempt Karate moves he cannot do. Only after Daniel demonstrates proficiency in the basic actions does the Karate master introduce challenging techniques.

The distance between how a novice begins and how an expert performs can deter both learning and teaching. But we cannot deny it as Daniel does, by trying to punch

right away or deny it as instructors by demonstrating how an expert performs. Miyagi overcomes this by assigning exercises that Daniel can do and that build toward skills that he cannot. His problems are both structured and open, providing practice, not just drill.

Similarly, this course begins with an idea students use every day, a *class* or category. Without the ability to use classifications students would not be able to recognize dangers, plan meals, or know where to socialize. A *class* is also a fundamental concept in object-oriented programming. To complete their projects students must be able to create and use classes in programming. Giving a *class* specific values creates an *instance*. *Classes* also service other *classes* using *methods*.

A short (10 min or less) demonstration introduces concepts followed by a first problem that requires students to use a Java editor to convert a source code into an executable code. Similar weekly problems and homework assignments follow that require students to use previous knowledge in new ways. Project requirements and past obstacles to student learning drive the problem design.

The instructor introduces three types of problems; (1) small structured exercises done in class with the coaching of the instructors, teaching assistants and undergraduate tutors; (2) weekly homework problems introducing new challenges solved outside the classroom by individuals; and (3) the final complicated computer game project executed both in and outside class. Structured problems help students face and overcome the shock of PBL. Learning by doing is threatening and practicing is not rewarding unless students are successful at initial problems.

By the second week, student teams create an initial set of specifications for their game. The prospect of taking charge of learning is exhilarating and daunting. Many students jump into the project with enthusiasm—brainstorming, proposing, and discussing ideas. Others cautiously wait for the instructor to tell them what to do. The instructor sets the structure, rules and principles of the class, supervises other coaches, grades the final project and provides feedback and help throughout the semester. The instructor's goal is to create a problem space that triggers the students' use of prior knowledge, search for information and construction of new mental models (Evensen and Hmelo-Silver 2000). Students have to gather information, investigate solutions, discuss experiences and reflect on new skills and knowledge (Savery 2006, p. 16).

The teaching interns are undergraduates who have completed the course and undergone PBL tutor training. These mentors (Teaching Interns or TI's as we call them) provide vital support. As role models, they help teams that don't believe that the instructor will actually allow them to design a game or that they have the ability to do so by sharing their past experience in the course. Able to speak "...the language of the students, using the concepts they use and explaining things in ways easily grasped by students," they provide *cognitive congruence* (Schmidt and Moust 1995, p. 709). They stand at the next step (zone of proximal development) in the students' learning ladder (Vygotsky 1986). The teaching assistant is a graduate student in charge of grading exercises and offering suggestions and feedback to the students.

The course began with a traditional 50-minute period three times a week format. PBL strained that schedule since there was not enough time for students to set up and solve problems. As a result the instructor adopted a new schedule of one three-hour workshop per week. The format required less set-up time and allowed many different learning activities to occur—problems, demonstrations, quizzes with discussion, and team project work. Computers in the classroom became tools and not toys.

## ***Phase 2: Project Application***

The transition between phases occurs at different times for different students as they make the leap from comfort zones to take on responsibility for their own learning. Some clamour for more team work time to design and execute the project. Other students wait until late in the course to act independently. This is a boon, as it allows students to advance at their own pace while providing positive models for laggards.

In the second half of the semester students face the final project in a do or die confrontation, realizing that, “we are going to make it or not”. They have to put the pieces together and make them work. Some students are shocked that the “teaching” is finished since they know everything they need. There are no more exercises, quizzes, or any other activity besides work on their own projects. Now they are programmers, the instructor is a senior expert, and the teaching assistants and interns are colleagues.

For example, students may want to make the characters in their games move at different speeds according to the terrain. Often they want them to jump or walk in novel ways (such as “moonwalk”). They must apply what they have learned in new ways or find new functionalities of the Java language that can solve their problems. As students encounter the difference between a *procedure* and a *function* in order to make progress on the big project, they must revisit earlier problems.

At the end of the class the students talk about the elegance and attractiveness of their programs, not about grades. What once seemed impossible becomes a doable task. The last week of the course explodes in celebrations more like those of triumphant athletes than the numb relief of students finished at last.

## **Conclusion**

The history and success of PBL in medical schools suggests that changing the values of numerous variables is necessary to achieve significant improvement. But many changes at once render an experimental design impractical. Each component presented its own problems of application and student acceptance. A case study approach allowed us to focus on problems of implementation. Formative assessment in the form of student feedback and continuous improvement was the key. In



addition, students filed weekly learning journals and student quality teams compiled suggestions for improvement.

Most evaluation techniques—quizzes, examinations, papers—work well as practice and feedback. They indicate errors and tell students they need to change their assumptions. They don't tell what students have learned. In this course, the final summative evaluation is the students' performance—how well they executed the design of a computer game based on the programming language they have mastered. That may be difficult to quantify, but not to know. Since the instructor worked with every team weeks before the final test performances of the games, he understood the strengths and weakness of each design. Further he knew in detail how each team member contributed to the over-all project. Assigning final grades was perhaps the easiest task in the course. Grade averages (up 5 points with smaller standard deviations) are proxies for the rate of completion and quality of the games. Over time, the final projects have become more imaginative, complex, functional, and aesthetically pleasing. At the same time student evaluations of the course and the instructor have improved after an initial dip with the change to PBL.

Ordinary indicators poorly measure PBL learning outcomes (Sanson-Fisher and Lynagh 2005). We believe that new methods are required. Future plans are to use the extended library of more than 40 games to create detailed and precise rubrics. By employing outside experts to devise rating scales based on random samples, we can assign numerical values to individual projects. Those values can record class achievements over time.

Movies like *The Karate Kid* wonderfully improve reality. In 1 month Daniel defeated his former tormentors. But the film's ending is true in the sense that the final test of any learning is performance. Miyagi, the master, risked failure, by setting a demanding goal in the face of Daniel's scepticism. He took responsibility for the outcome. We take that as the film's final lesson—if the student hasn't learned, the teacher hasn't taught (Nater et al. 2005).

## References

- Carroll, J. M., & Rosson, M. B. (1987). Paradox of the active user. In J. M. Carroll (Ed.), *Interfacing thought: Cognitive aspects of human-computer interaction* (pp. 80–111). Cambridge: MIT Press.
- Dewey, J. (1938). *Experience and education*. New York: Macmillan.
- Dweck, C. S. (2006). *Mindset: The new psychology of success*. New York: Random House.
- Evensen, D. H., & Hmelo-Silver, C. E. (2000). *Problem-based learning: A research perspective on learning interactions*. Mahwah: Erlbaum.
- Fischer, K. W., & Rose, L. T. (2001). Webs of skills: How students learn. *Educational Leadership*, 59(3), 6–12.
- Kilpatrick, W. (1918). The project method. *The Teachers College Record*, 19(4), 319–335.
- Nater, S., Gallimore, R., Walton, B., & Sinegal, J. (2005). *You haven't taught until they have learned: John Wooden's teaching principles and practices*. Fitness Information Technology.
- Ornish, D., Brown, S. E., Scherwitz, L. W., Billings, J. H., Ports, T. A., McLanahan, S. M., et al. (1990). Can lifestyle changes reverse coronary heart disease? The lifestyle heart trial. *Lancet*, 336(8708), 129–133.



- Sanson-Fisher, R. W., & Lynagh, M. C. (2005). Problem-based learning: a dissemination success story? *Medical Journal of Australia*, 183(5), 258.
- Savery, J. R. (2006). Overview of problem-based learning: Definitions and distinctions. *The Interdisciplinary Journal of Problem-based Learning (Spring)*, 1(1), 9–20.
- Schmidt, H. G., & Moust, J. H. C. (1995). What makes a tutor effective? A structural equations modeling approach to learning in problem-based curricula. *Academic Medicine*, 70(8), 708–714.
- Vygotsky, L. S. (1986). *Thought and language* (Translation newly rev. and edited ed.). Cambridge: MIT Press.

Innovative Practices in Teaching Information Sciences  
and Technology

Experience Reports and Reflections

Carroll, J.M. (Ed.)

2014, VIII, 238 p. 30 illus., Hardcover

ISBN: 978-3-319-03655-7