

# Learner Reflection in Student Self-assessment

Judy Kay, Lichao Li and Alan Fekete

School of Information Technologies  
The University of Sydney, NSW 2006, Australia

{judy, lli1, fekete}@cs.usyd.edu.au

## Abstract

Learner reflection is critical to effective, deep, transferable learning, especially in cognitively demanding areas, such as learning programming. This paper presents Reflect, a programming education system, which aims to facilitate student self-assessment and promote learner reflection through scrutable learner models. Reflect sets tasks and presents a set of example answers to read and assess. We report students' use of the system over one semester. Overall Reflect appears to help students understand the teacher's goals for the tasks set and to refine their own work in relation to these goals.

## 1 Introduction

Learner reflection is “a generic term for those intellectual and affective activities in which individuals engage to explore their experiences in order to lead to a new understanding and appreciation” (Boud, Keogh et al. 1985). There is a large body of evidence suggesting that learning effectiveness can be enhanced when learners pay attention to their own learning experiences by reflecting on the state of their knowledge and the learning process (Schön 1983; Boud, Keogh et al. 1985; Schön 1987). Learner reflection is especially important for cognitively demanding learning topics, such as learning programming. Schön identified three types of learner reflection, namely “reflection-in-action”, “reflection-on-action” and “reflection-on-reflection” (Schön 1983; Schön 1987), that may occur in any learning experience. We explore ways to support the first two of these in an intelligent programming education system named Reflect.

There are several examples of systems that capture a model of the learner<sup>1</sup> and make it available to support learner reflection: for example, the skill meter in ACT programming tutors (Corbett and Anderson 1995), the course progress chart in ELM-ART (Weber and Brusilovsky 2001), as well as in (Bull and Pain 1995) and (Hartley and Mitrovic 2002). Our system is unique in several aspects. First of all, Reflect aims to promote learner reflection through showing the learner a model of their progress in the process of student self-assessment.

Copyright © 2007, Australian Computer Society, Inc. This paper appeared at the Ninth Australasian Computing Education Conference (ACE2007), Ballarat, Victoria, Australia, January 2007. Conferences in Research in Practice in Information Technology, Vol. 66. Samuel Mann and Simon Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

Student self-assessment refers to “the involvement of students in identifying standards and/or criteria to apply to their work, and making judgments about the extent to which they have met these criteria and standards” (Boud 1991). It is generally acknowledged that student self-assessment contributes to the development of meta-cognitive skills, in that it can help learners develop the capacity to identify their strengths and weaknesses and direct their study to areas that require improvement. (Boud, Keogh et al. 1985) Such learning skills often characterize effective learners (Schön 1983; Boud, Keogh et al. 1985; Schön 1987). Student self-assessment has two key elements: the learner's identification of criteria or standards to be applied to their own work and the making of judgments about the extent to which work meets these criteria.

To make the student's self-rating in line with the teacher's judgment of the student's work, explicit criteria for satisfactory and unsatisfactory performance need to be established (Boud 1989). Teachers should design learning tasks to meet learning goals. In Reflect, the teacher makes these explicit by defining evaluation criteria for each task, and students need to self-assess themselves with criteria supplied by teachers. Although it is common practice in classroom teaching to encourage students to review grading criteria (Fekete, Kay et al. 2000), a system like Reflect is novel. Another distinctive feature of Reflect is that it lets students study examples and assess them according to criteria.

The work reported in this paper was conducted in the context of learning programming in C in an undergraduate subject, Software Construction I, which is a second year programming course that aims to teach programming in C in a UNIX environment. Reflect is one of the learning support tools designed to help students improve their learning and helping them pass the course. The system has evolved considerably since it was first built in 2002. Its original objectives were to enable students to self-assess their knowledge, monitor their learning progress and judge if they have achieved the required learning outcomes. This paper describes the latest version of Reflect with a learner modelling component that supports more intelligent feedback on learner's progress, thereby promoting learner reflection more effectively.

The following section gives an overview of Reflect. Section 3 describes the way that students used the system and Section 4 has further work and conclusions.

## 2 System Description

<p><b>Counting Elements</b></p> <p>You are required to iterate through a singly linked list of nodes and return the number of occurrences of a value. The lists data structure is provided below:</p> <pre>typedef struct node {     int n_value;     struct node *n_next; } Node;</pre>	<p>This task aims to improve your understanding of the following learning objectives:</p> <ol style="list-style-type: none"> <li>1. Coding Style - Good Use of Indentation</li> <li>2. Coding Style - Useful Identifier Names</li> <li>3. Coding Style - Comments</li> <li>4. Similar concepts in both C and Java - Control Flow</li> <li>5. Dynamic Data Structure - Linked List Traversal</li> </ol>
<p><b>Step 1: Examples</b></p> <p>Read and assess example solutions:</p> <p><a href="#">Example 0</a>  <a href="#">Example 1</a>  <a href="#">Example 2</a></p>	
<p><b>Step 2: Fill in the routine body</b></p> <pre>int count(Node *start, int val) {     }     }</pre> <p>Alternatively, you can <b>upload</b> your solution here (The texts in the above textarea will be ignored if you upload a file):</p> <input type="text"/> <input type="button" value="Upload"/>	
<p><b>Step 3: Self-assess</b></p> <input type="button" value="Save and assess"/>	

**Figure 1. Problem Statement**

To accomplish our research goal in the context of teaching programming in C in UNIX, domain knowledge of the subject was analyzed and learning objectives as well as students' common misconceptions (Fekete, Kummerfeld et al. 2003) were identified and summarised. Based on this information, a pragmatic learner model was defined. This was used to create a model for each student to monitor their knowledge state and learning progress. Students' interactions with the system are recorded in their learner models. Thus, these models always hold the system's beliefs of how well the students are performing in the subject. Making this information available to students by scrutinizing their user models, can help them become aware of their learning progress, thereby promoting learner reflection (Zapata-Rivera and Greer 2001).

Reflect has a collection of tasks for students to self-assess. As Figure 1 shows, there are three steps in the student self-evaluation process:

1. Students read and assess example solutions provided by the teacher;
2. They provide their own solutions to the problem and;
3. They self-assess that solution using criteria the teacher has defined for that task.

On the problem statement page, at the upper right in Figure 1, there is a list of learning objectives that the teacher would want students to learn from the task. The system keeps all these learning objectives in a dictionary, which represent the domain knowledge. When reading and assessing example solutions as well as writing their own answer to the problem, students need to pay special attention to such learning objectives. For example, in Figure 1, the task is intended to help students learn about

1. Coding style: Good use of indentation, useful identifier names and comments;

2. Similar concepts in C and Java: Control flow and;
3. Dynamic data structure: Linked list traversal.

To begin the self-assessment process, students are required to read and assess all examples the teacher provides. These examples are not necessarily 'perfect' solutions. Rather they provide opportunities to explore interesting and important ideas associated with the learning goals. We usually create these examples, which demonstrate common misconceptions, poor style and similar elements, after grading final exam questions. We also strive to provide multiple examples, so we can illustrate different ways to do one task.

Students also need to assess the example solutions they read with criteria provided by the teacher, as in Figure 2. Each criterion corresponds to one of the learning objectives of the task. Consequently, each criterion asks the student to rate one key aspect of the example solution. For instance, in Figure 2, it asks about the coding style of the solution. The student can choose from a range of options for each marking criteria, for example, excellent, good, ok, poor or rotten. These example solutions have been pre-assessed by teachers using the same criteria. Students' assessments are then compared to the teacher's assessment, as illustrated in Figure 3. The comparison gives the student feedback on:

- How the teacher marked the example;
- The difference between the teacher's assessment and the student's assessment and;
- Why the teacher assessed it that way.

Since the marking schemes used to assess the solution are consistent with the learning objectives of the task, the

discrepancy between the student's and the teacher's assessments indicate how well the student comprehended these objectives. The probability that the student has understood each learning objective is then estimated and recorded in his/her learner model, updating the system's belief of how well he/she understands that objective. This information is used to illustrate the student's learning progress to promote learner reflection-on-action, as upon seeing this information, the student may explore why the teacher thinks differently from him/her and learn how to think as the teacher does, so to achieve a new level of understanding of the issues.

**Assessing SOFT2130 Counting Elements example 0**

You should now assess the example using the following marking scheme. This window can be moved so that you can view both the example solution and the marking scheme at the same time.

Rate the style of the code, especially with respect to the use of sensible identifier names.	no opinion
Rate the style of the code, especially with respect to the good use of indentation.	no opinion
Rate the style of the code, especially with respect to comments presented.	no opinion
Rate the design of the loop, with respect to: <ul style="list-style-type: none"> <li>Sensible exit condition in loop.</li> <li>Non-recursive solution.</li> <li>Iterating the loop correctly.</li> </ul>	no opinion
Rate the overall correctness of the code.	no opinion

Done

**Figure 2. Solution Assessment**

**Comparing assessments for SOFT2130 example 0**

Criteria	Yours	Ours	Disc
Rate the style of the code, especially with respect to the use of sensible identifier names.	excellent	poor	3
Rate the style of the code, especially with respect to the good use of indentation.	good	excellent	1
Rate the style of the code, especially with respect to comments presented.	ok	ok	0
Rate the design of the loop, with respect to: <ul style="list-style-type: none"> <li>Sensible exit condition in loop.</li> <li>Non-recursive solution.</li> <li>Iterating the loop correctly.</li> </ul>	poor	excellent	3
Rate the overall correctness of the code.	rotten	excellent	4
<b>Total discrepancy</b>			<b>11</b>

Aim to minimise the discrepancy between your assessment and ours.

**Explanation of our assessment**

The variable name foo is not very descriptive of it's purpose.  
The comments do not increase the level of understanding of the code.

done

**Figure 3. Comparison of Student's (Yours) and Teacher's (Ours) Assessments of an Example and a Measure of the Discrepancy between Them**

Once the student has viewed and assessed all the example solutions of the task, they can attempt their own answer to the problem; submitting it in the large answer pane in Figure 1. Then, the self-assessment phase begins, shown as Step 3 in the figure.

Students assess their own solutions just as they did for the example solutions, using the same marking criteria, as shown in Figure 2. Importantly, when students self-assess their solutions to a problem, it encourages reflection-in-action (Schön 1983; Schön 1987). They are able to read each marking criterion and reflect, as they rate a solution with respect to the criterion. Since they have read and assessed the example solutions in Step 1, they can

rate their own solutions more accurately. As the criteria are shown in a new pop-up window, they are more likely to consider each criterion as they review their own solution. For instance, the second element of the fourth marking criterion in Figure 2 explicitly requires students to supply non-recursive solutions, students may realize that they had used a recursive solution, which is not preferred or encouraged in our subject. Because these criteria are created by teachers to be consistent with the teaching goals, when students assess their solutions with the set of marking criteria, they will be helped to reflect and learn, focusing on the relevant learning objectives. After self-evaluation, the students' solutions and assessments are saved, so that they can later reassess the solution. This concludes the three steps of student self-assessment process in Reflect.

Reflect also provides each student with an individualized user profile, which displays the student's learning progress on a concept-by-concept basis in Scrutable Inference Viewer or SIV (Uther 2001; Kay and Lum 2005). SIV displays a user model and allows it to be viewed in different ways. On the profile page, next to SIV, there is also a list of hyperlinks for each learning objective. If the student clicks on one of them, more statistics about the concept are presented in a popup window, as shown in Figure 4. This information includes the student's mark for the concept, the number of times the student has attempted it, the class' average mark for it and the tasks that have it as a learning goal.

The user profile promotes reflection-on-action (Schön 1983; Schön 1987) as it shows students a visualisation of their learning progress by externalising their learner models with SIV (Zapata-Rivera and Greer 2001). It allows a user model to be explored in certain ways, such as selecting, deselecting and expanding a user model component and displays the model accordingly. In the SIV display, teaching/learning goals are displayed with different indentations and in different colours. In particular, it answers the four questions, which should be addressed by a scrutable learner model, mentioned in (Kay 1997), as it shows

- What they know (the learning goals that are displayed in green in SIV);
- How well they know it (the saturation of colours, if a student knows a concept better, the concept is displayed in SIV in darker green);
- What they do not yet know (the learning goals that are displayed in red or yellow in SIV) and;
- How to learn it (what tasks aim to teach the concept, e.g. in Figure 4, the student can do *Task 3* to learn about *Linked List Traversal*).

From this information, students are encouraged to think about what they have learnt and how they have learnt it, i.e. reflecting on their experience. Furthermore, students are able to compare the system's beliefs about their C programming knowledge with their own beliefs. In this way, reflection is further encouraged; especially if the system's beliefs and the student's own beliefs differ. SIV can also show students what and how the learning

concepts are related by displaying them in different sizes, with larger ones more closely related to the currently selected concept. It then becomes possible for students to improve their understanding of one concept by working with closely related concepts.

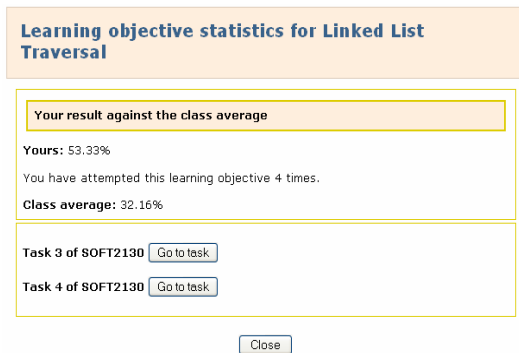


Figure 4. More Useful Statistics

In summary, both learner reflection-in-action and reflection-on-action, facilitated by scrutable learner modelling are supported by the process of assessing solutions, reviewing overall progress with SIV and reviewing detailed performance comparison against the class.

### 3 Evaluation

In this study, a general overview of students' system usage is presented and a selected group of students' submitted solutions were reviewed in detail qualitatively to reflect their underlying learning behaviour.

As described in Section 2, students interact with the system when they view and assess example solutions and when they write and self-assess their own solutions. Their assessments of example solutions were compared with the teacher's assessment. This was used to calculate a value indicating how well the student understands the learning objectives. Such values were saved and amalgamated over time to create individual learner models. The students' own solutions were also saved in the system.

Reflect activities have been integrated into the laboratory work in our C programming in UNIX subject. Students needed to complete Reflect tasks before or during weekly laboratories for course credit. Typically, each week, students were required to do one or two tasks. Students' motivation was increased when they became aware that the Reflect tasks were based on past exam questions. Our records show that the system was used by most of the class. Furthermore, it is to be noted that although completing Reflect tasks was part of the subject's assessment, there were still students who did not use the system as instructed, partially because credit associated with the system was about 5% of the total assessment. Nevertheless, many students used Reflect as an extra learning resource in addition to the lectures and tutorials they attended, as shown from our analysis of the learner models and system logs.

There were over 260 students enrolled in the course in 2005. Forty-six students' activities were analysed in detail. Of these 46 students, 20 earned final grades at the top of the regular class, and 26 were borderline students (they either just passed the subject or just failed). They were chosen because these are two interesting populations to study. Table 1 gives an overview of the overall activity of these students. It shows the average number of tasks each student attempted, the average number of solutions he/she provided (where multiple solutions could be submitted), the average number of self-assessments he/she made as well as average the number of evidence the system gathered from each student. Fifteen tasks available, with thirteen set as part of the subject's assessment. The top 20 students used the system much more than the borderline students. Even so, only about 60% of the top students used the system. Notably, one borderline student used the system very frequently, and the system gathered 266 pieces of evidence about this student. Excluding this student, the average number of evidence gathered from borderline students was 25.7.

	Tasks attempted (max =15)	Solutions provided	Solutions self-assessed	Evidence gathered
Top 20 students	8.3	10.8	8.6	60.5
Borderline students	3.7	5.04	3.04	34.9

Table 1. Summary of Students' System Usage

To gain a better understanding of students' learning behaviour when using Reflect, we conducted a detailed qualitative evaluation of the actual solutions that students submitted. Sixteen students, with whom we had personal contact during the semester, were chosen. We selected students we knew to be hard working during the semester and likely to make serious attempts at practical work. These students are representative of a broad range of performance levels with examination marks range from the top of the class (81/100, the pass barrier mark is 40) to the bottom (17/100). Eight of them passed the exam, and the other eight failed. In this study, these students' answers to four Reflect problems were reviewed. The four tasks are List Traversal and Counting Elements, which are two linked list traversal problems, and Test Script and Fibonacci, which are UNIX shell scripting questions. They were chosen because dynamic data structures and UNIX concepts were two of the most important components of the subject curriculum and, thus, most students made attempts to do these tasks.

The List Traversal task requires students to write a function that goes through a linked list and find the average value of all the integers in the list. Since students had the opportunity to read our example solutions first, it is not surprising to find that their own solutions are influenced by our code. Unfortunately, sometimes they were too influenced by the incorrect answers, their answers were exactly the same as the supplied examples. Nonetheless, we found that more able students were more likely to provide a solution that was different from the

examples and they could identify errors in our examples (recalling from Section 2 that our example solutions are not “perfect”) and rectified them in their solutions. We call these “authentic” answers. For example, for the List Traversal task, eleven students submitted answers and eight of them passed the exam. Four students who passed the exam provided authentic answers. The task has two examples answers illustrating different misconceptions. The remaining seven students had solutions similar to the more correct example, which is presented below:

```
float meanAge(Node *start) {

    int sum, count;

    while (start) {

        sum += start->age;

        count++;

        start = start->next;

    }

    return (float) (sum / count);

}
```

This answer is mainly correct, but with a critical problem. The integer variables, sum and count, are not initialized and using un-initialized variables can lead to unexpected program behaviour. Our commentary on the examples explained this. Four other students who passed the exam had solutions that were similar to this example, but corrected the error. For example, one of them had the following answer:

```
float meanAge(Node *start) {

    float sum = 0;

    float count = 0;

    while(start != NULL) {

        sum = sum + start -> age;

        count = count + 1;

        sum = start -> next;

    }

    return sum/count;

}
```

This solution initialized the integer variables but still failed to handle the case when the variable “count” is zero. On the other hand, although the three less able students had solutions similar to this more correct example, they also reproduced the un-initialized variable error. It is apparent

that they were not able to recognize this error from our examples. There was also one student who submitted two solutions: the later submitted solution showed corrections and improvements to the first submission.

The Counting Elements task is similar to the List Traversal task. Students need to write a function that traverses a given linked list with each node containing a single integer value and count the occurrences of a particular value. There are three example solutions. One of them is correct. All fourteen students who submitted a solution were able to avoid the mistakes from the two incorrect solutions. However, four students (two passed the exam, two failed) submitted solutions that are similar to the correct example but incorporated new errors that were not present in the two erroneous examples.

The Test Script task in Reflect asks students to write an UNIX shell script for testing that is flexible and versatile. We provided three examples. Two of them were clearly wrong and one was correct. By studying the students’ submissions, we had similar findings to the linked list questions. Eight of thirteen students who provided a solution had authentic or solutions that were similar to the correct example. One top student submitted an authentic solution. Three failing students and one student with 80/100 in the exam copied the wrong example. This again shows that weaker students were unable to recognize the errors in the examples.

The Fibonacci shell scripting task incorporates UNIX concepts such as pipes and redirections. We provided two example solutions, one correct and one with serious mistakes. We observed that most students’ solutions were a copy of or very similar to the correct example (12 of 14 students who made submissions). Of the two remaining students, one who failed the exam copied the example with serious mistakes and the other who achieved the top mark (81/100) provided an authentic solution but with errors.

We also reviewed how students self-assessed their solutions and found that the more able students rated their own solutions more carefully. The self-assessments of their own answers were often similar to how a teacher would assess them. The less able students, on the other hand, tended to over-rate their solutions. This was in line with earlier studies from (Boud 1989).

To sum up the qualitative evaluation, we found that students were strongly influenced by the supplied code. However, it also shows that the approach of reading examples before writing a solution does necessarily enable students incorporate common elements from our code. The examples are beneficial to learning so long as the tasks are designed to ensure that straight copying is not attractive.

## 4 Discussion and Conclusions

So far, our experience with Reflect has been in a subject devoted to programming, and so all of the examples have been questions where the task is primarily synthesis. In other fields of computing, the nature of assessment tasks is often quite different, and this raises some issues for a self-assessment system such as Reflect. We next offer

some thoughts on how to use our system with theory material, from a data structures subject.

Traditional data structures classes use assessment with a large weighting on questions requiring tracing an operation. For example, students are asked to show how a binary search tree will be modified when a particular element is deleted. These questions are usually graded as correct or incorrect, without much consideration of partial marks, and so the self-assessment system is of limited use. If a correct answer is shown, the student will not have difficulty answering themselves. Thus the best use of Reflect is to show students purported solutions that result from common misconceptions (these are like appropriate distracters in multi-choice questions). Identifying that the answers are wrong can help students remove these misconceptions in their own thinking, and this process is assisted by the message justifying the lecturer's low score on "correctness", which is displayed after the student has rated the purported solution.

Another common type of assessment task in the data structures subject requires the student to explain or to prove something. Here it is easy to offer the student many interesting potential answers, each with a mixture of good and bad features. The difficulty for the instructor is to choose assessment criteria which apply sensibly to "answers" with a mistake that means the flow-on reasoning takes an incorrect path. Of course, this is also a difficulty when designing the marking scheme for conventional assessment questions. For example, in a question on collisions in hashing, many students are confused between open and closed hashing. With Reflect, it would be good to use a purported answer to build awareness of this error, but then how should one score a criterion like "explains how the algorithm re-establishes the data structure invariants after the operation"? We suggest having marking criteria which divide the ideas up more finely. In the example, one could define three criteria: "explains what invariants the data structure must satisfy" and "states that the data structure must be re-established after the operation" and "explains what invariants are violated after the initial changes" and "explains how the structure can be modified to re-establish the invariants". A side-benefit of such a detailed set of assessment criteria, is that it indicates to students a good format for presenting their own explanation.

One question type is very important for data structures, and can be treated in either of the previously mentioned ways. This is where students need to analyze the runtime cost of an operation. We feel that Reflect is best used if the question asks to give the expression for the cost and justify the answer. Here a nice use of Reflect is to define the marking criteria to include separate aspects for each of the main ideas that are needed in an answer. For example, one can have one criterion which is "does the answer show awareness that the operation acts on each level of the tree", and another which is "does the answer show awareness that the worst-case number of levels is linear in the number of nodes", and yet another which is "does the answer show awareness that an operation which does linear work in some cases, and constant work in others, has worst-case cost which is linear".

We are currently exploring improvements for Reflect. The first, and most pressing, is to provide automatic evaluation of a student's own solution. If students' solutions can be automatically evaluated and the testing results can be resolved and fed into the learner models, we can provide more concrete feedback, which should make this part of the system more valuable to students, possibly overcoming the overestimation problem in the current system. A second goal is to provide learning guidance. With the existing learner models, it is possible to adaptively guide students, based on their knowledge state and select the most suitable learning path for them. Since learner reflection mostly occurs when the learner feels challenged (Boud, Keogh et al. 1985), appropriate choice of tasks for the individual can may promote learner reflection, especially reflection-in-action. Furthermore, we want to investigate further the impact on students' learning behaviour if they read examples first, compared with writing the program first and then reading example solutions.

In summary, we have presented Reflect, a student self-assessment system with support for learner reflection. Two types of learner reflection, namely reflection-in-action and reflection-on-action are supported by allowing the student to self-assess solutions and by providing intelligent and informative learning progress feedback through open learner modelling. We presented a qualitative analysis of student use of the system. In light of our earlier studies, which have shown that students feel comfortable using the interface that displays the learner model (Uther 2001) and they value the reflection opportunity and experience in general, we believe that Reflect can provide motivated students with help that they value and the system measures give a good indication of student's ability to "see" code from the perspectives that the teacher intended that they consider for their current learning objectives.

## 5 References

- Boud, D. (1989). "The role of self-assessment in student grading." *Assessment and Evaluation in Higher Education* **14**(1): 20-30.
- Boud, D. (1991). "Implementing Student Self-Assessment." Campbelltown: Higher Education Research and Development Society of Australasia. HERDSA Green Guide **5**.
- Boud, D., R. Keogh, et al. (1985). Promoting reflection in learning: A model. *Reflection: Turning Experience into Learning*. D. Boud, R. Keogh and D. Walker. London, Kogan Page: 18-40.
- Bull, S. and H. Pain (1995). "Did I say what I think I said, and do you agree with me?" Inspecting and questioning the student model. *World Conference on Artificial Intelligence in Education*, Charlottesville, VA, AACE.
- Corbett, A. T. and J. R. Anderson (1995). "Knowledge tracing: Modeling the acquisition of procedural knowledge." *User Modeling and User-Adapted Interaction* **4**: 253-278.
- Fekete, A., J. Kay, et al. (2000). Supporting Reflection in Introductory Computer Science. SIGCSE.

- Fekete, A., B. Kummerfeld, et al. (2003). Identifying and improving the learning of borderline students in SOFT2004. Sydney, Australia, School of Information Technologies, The University of Sydney.
- Hartley, D. and A. Mitrovic (2002). Supporting Learning by Opening the Student Model. ITS 2002, Springer-Verlag Berlin Heidelberg.
- Kay, J. (1997). Invited keynote address: Learner know thyself: Student models to give learner control and responsibility. ICCE'97 International Conference on Computers in Education.
- Kay, J. and A. Lum (2005). Exploiting readily available web data for scrutable student models. 12th International Conference on Artificial Intelligence in Education, Amsterdam, Netherlands.
- Schön, D. A. (1983). The reflective practitioner. New York, Basic Books.
- Schön, D. A. (1987). Educating the reflective practitioner. San Francisco, Jossey-Bass Publishers.
- Uther, J. (2001). On the visualisation of large user models in web based systems. Basser Department of Computer Science. Sydney, The University of Sydney.
- Weber, G. and P. Brusilovsky (2001). "ELM-ART: An adaptive versatile system for web-based instruction." International Journal of Artificial Intelligence in Education(12): 351-384.
- Zapata-Rivera, J.-D. and J. E. Greer (2001). Externalising learner modelling representations. AI-Ed 2001 Workshop: External Representations in AIED: Multiple Forms and Multiple Roles. San Antonio, Texas.