Software engineering as a model of understanding for learning and problem solving

Paul Gibson and Jackie O'Kelly
Computer Science Department
NUI, Maynooth
Ireland

INTRODUCTION

- Observing young children in schools as they play with games and puzzles, and solve problems.
- Motivated by the need to try to get a better understanding of the algorithmic learning process in order to improve the teaching of software engineering.
- Using Problem-based Learning (PBL) as the environment in first year computer science.
- Motivated by the need to try to overcome the inherent difficulties first year students have with programming.

Learning Theories that have influenced our research

- Piaget – Cognitive Structure
- Bruner – Constructivist perspectives of learning
- Guilford – Structure of Intellect
- Gardner – Multiple intelligence
- Papert – Computers and Education
- Schoenfeld – Mathematics as problem-solving
- Bloom – Taxonomy of Educational Objectives

Software Engineering Cognitive Tools

Reflinement

- Refinement is a method of software construction that allows (abstract) specifications to be iteratively refined into other (more concrete) specifications resulting in an efficient implementation.
- Algorithm (process) refinement allows this translation to occur over a procedure or code fragment.
- Data refinement allows the data representation to be translated into a better data structure.

Subclassing: Extension

- Adding an extra state to a system, so that we extend its behaviour. The new system (subclass) does everything the old system (superclass) does, and more.
Subclassing: Extension

- Adding an extra state to a system, so that we extend its behaviour. The new system (subclass) does everything the old system (superclass) does, and more.

Subclassing: Specialisation

System (a) The Stop, Go signals alternate and traffic is always flowing, in only 1 direction at a time.

System (b) Where the traffic is either stopped in both directions or flowing in both directions.

There is no way for the system to move between these 2 separate sub-behaviours. The behaviours of system (a) and system (b) are specialisations of the original system.

Re-Use through Composition

- Existing software artefacts
  - set of documents and models that are built during the engineering of a software system
  - Analysis, requirements, validation, design, verification, implementation, tests, maintenance, versioning and tools.

are reused, in a wide range of ways, in the construction of a ‘new’ software system.

Genericity: Universal

- In software engineering, modelling languages usually provide a means of specifying parameterised behaviour. Typically we see this in the form of a generic data structure.

- The STACK with ‘LIFO’ behaviour offered through push and pop is a classic example.

- If there exists a generic stack then all you need to do to reuse this, is instantiate the type parameter.

Genericity: Constrained

- Sorting: we should be able to sort lists of any type of entity provided, there is a way of comparing and ordering any 2 such entities. In effect, we are constrained to sorting only those things that can be sorted.
Re-using or re-usable

In the traffic light example it would be natural for a software engineer to:

1. Transform the state into a composition of 2 boolean variables in order to re-use an already existing well-understood component,
2. Make the traffic light system a re-usable component for re-use in larger, more complex systems.

Case Studies

Case study 1 – Searching (in schools)

Run over a 7 year period, session run 16 times with 16 different classes in 10 different schools.

Age profile: 6 – 17 years (mean age 13)

Average class size: 18

Phase 1

Pre-requisite: Confirming that all children are able to compare the lengths of pieces of string and match those of the same length.

Phase 2

Hide a number of pieces of string in a number of boxes (one per box), hand the children a piece of string and ask them to find the matching string in one of the boxes. However, they are told that they can only open one box at a time, and that when a box is closed it must contain the piece of string that was in it originally.

Process refinement

Never look in a box that was already looked in.

Data Refinement

Boxes searched in an ordered fashion (left to right)
Boxes already searched are marked with a pencil.
Boxes moved from a 'not yet examined pile' to an 'examined pile'.

Phase 4

Order the boxes based on the length of strings within, before asking the children to play the game.
Over a period of time the children effectively refine their solution to a binary search.

Phase 5: Generalising the problem

The strings are replaced by some other physical entity.
All children manage to generalise the solution for strings to other physical entities which can be ordered in some intuitive fashion.

(This is an example of constrained genericity)
Phase 6: Working with abstraction

Play guess the number (0 – 100), by asking questions to which only the answers yes and no are allowed.

Some of the children employ the same algorithm for the ‘guess the number’ as they did for ‘find the string’.

Phase 7: Observing compositional re-use and subclassing (specialisation)

- The number game is altered such that there are 2 numbers that add up to 100, find them both.
- The original binary search.
- The calculation of the largest (missing number) as a simple subtraction.
- A symmetry in the reformulation of their approach in terms of finding the largest (not smallest) element first.

Case study 2: The Tower of Hanoi (University)

- Run over a 2 year period, session run once with 38 different groups.
- Age profile: 18 – 23 years
- Average group size: 5 - 7

- All groups identified the following:-
  - Facts: 3 pegs, n discs of decreasing size
  - Constraints: Move only one disc at a time, cannot place bigger disc on to a smaller disc.

- Problem given to students in a Problem-based Learning workshop, in the first week of term.
- Half the groups were given a prop.

No prop category:-(data refinement and abstraction)

- Discussed different ways to represent the pegs and the discs,
- Investigated the use of chairs, people and books
- Put a label on the discs and the pegs.

Constrained genericity – solving the problem using different entities provided there was a way of comparing these entities in terms of size (larger, smaller)
Plot category

$\text{Relied on the prop to solve the problem, apart from one group who believed the prop was designed incorrectly.}$

$\text{They identified a “solution” incorporating a circular pattern, where the moves were refined to be clockwise and anti-clockwise.}$

Solve the problem with the same constraints in less moves?

$\text{Reduced problem to the simplest instance (1 disc), solved it in two different ways, accepted the solution with the least moves.}$

$\text{Increased complexity of problem by adding another disc ……..}$

Identified a pattern for even numbers and a different pattern for odd numbers.

Hypothesised and confirmed their hypothesis for the optimal solution for $n$ discs.

(1) $\text{moves for } n = (\text{moves for } n-1) \times 2 + 1$
(2) $2^n - 1$

Software engineering techniques: -

$\text{Process refinement}$
$\text{Subclassing and genericity}$
$\text{Composition}$

Comparison

$\text{School children (6 – 17 years) and University students (18 – 23 years) demonstrate an implicit understanding of software engineering techniques and algorithmic understanding.}$

$\text{There are differences in their communication abilities.}$

$\text{The size of the groups is different}$

$\text{The time for the sessions is different.}$

$\text{The maturity of the students is different.}$
Conclusions

§ We observed little difference between children and adults in how they learn about computation.

§ The common framework is usefully modelled using well-understood software engineering techniques.

Future Work

§ Develop a *theory of problems* that could be used in reasoning about the order in which problems should be presented to CS1 students, and the relationships between these problems.

§ Implement a collect of Java applications for objectively gathering data about the way in which school children solve problems.

§ Feedback what we have learned from this collaborative research into the sessions we run in schools and with university students.

§ Consolidate our proposal for the theory that *software engineering provides a good framework for reasoning about how children and adults learn to solve problems.*