Application of Test Informed Learning with Examples (TILE) in existing programming courses

Introduction

Teaching programming requires finding effective ways to convey complex concepts to students. Test Informed Learning with Examples (TILE) is a new approach that builds on the principles of TDL [1]. It introduces software testing in introductory and advanced programming courses in early, seamless, and subtle ways. TILE makes testing an inherent part of programming, rather than a separate activity, by introducing it from the very first example program and using clever and indirect methods to teach testing knowledge and skills.

TILE Types

TILE has different types, including Test Run TILEs, Test Cases TILEs, Test Message TILEs, and Test Domain TILEs. Test Run TILEs can be made by asking students to test the program or assert the right output instead of asking them to run the program. Test Cases TILEs can be created by providing students with more concrete examples of possible test cases that they should use to check their code. These examples should challenge their code on some corner cases and other less happy tests, rather than just the happy path execution. Test Message TILEs hide a subliminal message about the importance of testing, while Test Domain TILEs require more creativity than the previous ones and use examples from the testing domain directly, rather than from a well-known domain.

Some example assignments

Here we provide one assignment of each type of the approach. The assignments have different levels of difficulty and cover different programming concepts. They can be used in existing courses but are also created to show the possibilities of using TILE with existing assignment. Each example ends with some information about the testing aspects that are integrated in the assignment.

A *test run* TILE assignment

Make a program that receive values for three variables `a`, `b` and `c`, and interchange their values as follows:

- `b` takes the value of `a`,
- `c` takes the value of `a`, and
- `a` takes the value of `c`.

This must be done WITHOUT using auxiliary variables, that is, additional helper variables that are not a, b or c, and are used to store some values.

The execution of the program should result in the following:

```
>>> %Run
  Enter the value of the variable a: 4
  Enter the value of the variable b: 2
  Enter the value of the variable c: 7
  The value of a is 7
  The value of b is 4
  The value of c is 4
```

Execute tests through the console and check the output. Does your program work for negative numbers? Does it work for characters? Does it work for reals? Can `a`, `b` and `c` have different types? Should your program work for all these cases?

This exercise was TILEd by adding the last paragraph. We explicitly ask the students to test for different types of values. Most students, because of the example execution convert the user input to 'int', but that is not necessary for the swapping, anything can be swappped. Asking them to test with all kinds of values makes them aware of the assumptions they made when reading the exercises and hence how testing is good to find errors.

A *test cases* TILE assignment

Implement a program that reads three integer values: day, month, and year of a person's birth. Using this data, the program should show a four-digit PIN associated with the date of birth. The PIN is calculated as:

- p1 = (d1 + d2) % 10.
- p2 = (m1 + m2) % 10.
- p3 = (y1 + y4) % 10.
- p4 = (y2 + y3) % 10.

For example, if the date entered is 29 9 1975, the PIN would be 1 9 6 6:

- p1 = (2 + 9) % 10 = 1.
- p2 = (0 + 9) % 10 = 9.
- p3 = (1 + 5) % 10 = 6.
- p4 = (9 + 7) % 10 = 6.

```
>>> %Run
  Enter your day of birth: 29
  Enter your month of birth: 9
  Enter your year of birth: 1975
  Your PIN is 1 9 6 6
```

test case ID	inputs			expected output (PIN)
	day	*month*	*year*	
1	10	12	1522	1 3 7 2
2	1	1	1	1 1 1 0

test case ID	inputs			expected output (PIN)
3	27	3	1978	9 3 9 6
4	55	28	300	0 0 0 3
5	356	903	1568	1 3 9 1

Look at test cases 4 and 5. Are they valid? Inputs 55 and 356 are not valid numbers for a day of birth. However, our program works and calculates a PIN. Our programming language does not know what birthdays are and when they are valid. For the programming language, the 3 inputs are simply whole numbers. If we want our program not to calculate a PIN when the date is not valid, then we should add conditions that verify the inputs.

A table with test cases was added and the students were made aware of the test cases that not really contained valid dates but still calculated a PIN number.

A *test message* TILE assignment

Mad Libs is a phrase template word game where a player asks others for a list of words to substitute for blanks in a story, often comical or nonsensical, and which will be read aloud later. We are going to make a little Mad Libs.

Look at the following example:

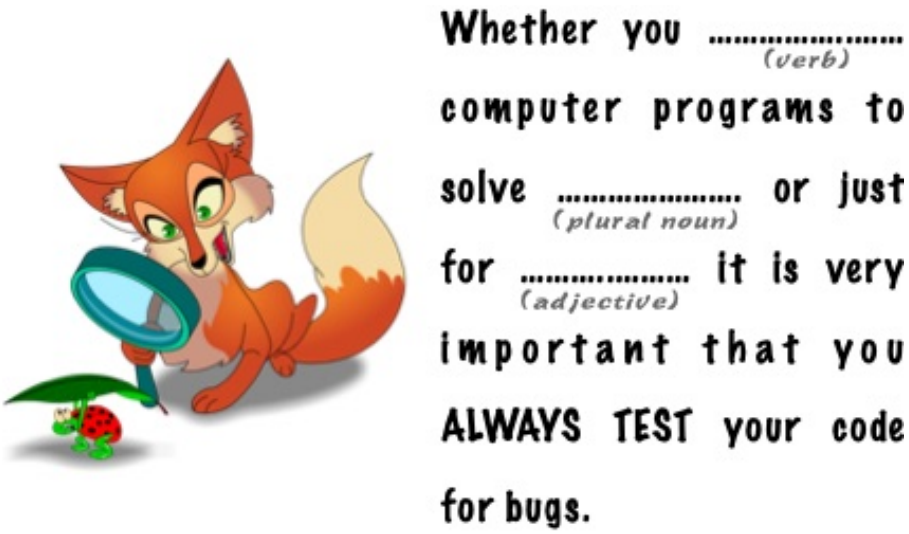

Figure 1: MadLib example

We need to ask the player for the following words in English:

- verb, for example: `write`

- plural noun, for example: `problems`
- adjective, for example: `fun`

So for these examples, our program returns:

```
Whether you write computer programs to solve problems or just
for fun, it is very important that you ALWAYS TEST your code for
bugs.
```

Try other inputs and try to come up with a funny phrase.

This TILE contains the message that testing is important.

A *test domain* TILE assignment

Imagine you just wrote a program that sorts a list of numbers and evidently now you need to test it. Write a program that helps you test your software. The program should ask you whether you took care of all the special cases and you answer yes or no. At the end, the program tells you how well you did. Below is an example of the output your program should produce. Try to extend the possible cases with other important things to test (there are at least two that are useful to add).

```
Hello! I'm gonna help you improve your sorting program!

Did you check a basic case like:
 [3, 1, 8] is sorted into [1, 3, 8]? (y/n) y
Excellent!

Did you check what happens when the list is empty? (y/n) y
Nice.

Did you check what happens for a list with a single
element, like [3]? (y/n) y
Well done!

Did you verify it also works with negative numbers,
like [4, -8, 10]? (y/n) n
You'd better try that right now!

That was my last question!
You took care of 75% of the cases. Well done!
```

The TILE Repository

We created a repository with over a hundred TILE exercises that can be used by educators to incorporate into their courses. This provides several benefits for educators:

- Variety of exercises: With a repository of this size, educators have access to a wide variety of exercises, which they can use to create a diverse set of assignments and assessments. This helps to keep students engaged and motivated, as well as provide opportunities for differentiation.
- Time-saving: Educators do not need to spend as much time creating new exercises. This saves them time and allows them to focus on other aspects of teaching, such as lesson planning and student support.
- Quality control: The repository is created and curated by a team of experts to ensure that the exercises are high-quality and aligned with educational standards. Educators can therefore trust that the exercises they select from the repository are well-designed and appropriate for their students.
- Collaboration: The repository facilitates collaboration among educators, as they can share and contribute their own exercises to the repository, building a community of practice and fostering a culture of continuous improvement.

Not only does the repository contain the exercises, each exercise is annotated with meta-data to match exercises with the requirements of the course in which one wants to incorporate the exercise. The meta data contains information related to the audience, learning goals, pre-requisites and the programming topics of the exercises.

The repository can be found at https://tile-repository.github.io/.

Conclusion

TILE is a promising approach to introduce testing in introductory programming courses. It allows for the seamless integration of testing in programming exercises, making students more aware of the importance of testing and helping them develop the skills and knowledge necessary to write testable code. By using clever and indirect methods to teach testing, TILE ensures that testing is introduced in a subtle way that does not overwhelm students or detract from the main focus of the course. With its four different types of TILEs, TILE provides instructors with a flexible and adaptable approach to teaching testing, allowing them to choose the type that best suits their needs and the needs of their students. The repository of TILEd exercises provides time-saving ways for teachers to re-use existing, well designed programming exercises which have already been TILEd. The repository provides a way for educators to benefit and contribute to the TILE community.

Overall, TILE has the potential to significantly improve the way testing is taught in introductory programming courses and to help students become better programmers.

References

[1] Janzen, D.S. and Saiedian, H. 2006. Test-driven learning: Intrinsic integration of testing into the CS/SE curriculum. *SIGCSE Bull.* 38, 1 (Mar. 2006), 254–258.