Enseñanza temprana del testing en cursos de programación

Tanja E.J. Vos y Beatriz Marín
{tvos, bmarin}@dsic.upv.es DSIC
Universitat Politècnica de València
Cno. de Vera, s/n, 46022, Valencia

1. INTRODUCCIÓN

Se ha demostrado que la integración de la educación de testing en los cursos de programación trae beneficios en el proceso de aprendizaje, como la mejora del desempeño de los estudiantes, ya que reciben retroalimentación acotada a su desempeño a la vez que el proceso de evaluación es más objetivo [1]. Sin embargo, la integración de la educación de testing en los cursos de programación tiene todavía muchos traspiés, como por ejemplo los identificados en la revisión sistemática realizada por Scatalon et. al. [1] que se listan a continuación:

- Se realizan ejercicios muy simples debido a la falta de capacidades de testing de los estudiantes. Testear es una habilidad que requiere conocimiento de programación que los estudiantes están recién adquiriendo en los cursos de programación. Por este motivo, los estudiantes no pueden escribir buenos casos de prueba aún. Esto es conocido como el problema del huevo y la gallina pues necesitan conocer de programación para crear buenos casos de prueba, pero necesitan saber de testing para crear buenos programas.
- Los estudiantes tienen una actitud negativa hacia el testing a pesar de que reconocen su importancia.
- Los cursos de programación ya están planificados, por lo que integrar tópicos de testing significa agregar contenido adicional en la misma cantidad de horas lectivas.
- Es necesario hacer trabajo adicional para ajustar el material de los cursos de programación para incorporar contenidos de testing en el material entregado a los estudiantes.

Para resolver o minimizar estos problemas, en este trabajo se presenta la propuesta TILE (por sus siglas del inglés Test Informed Learning with Examples: aprendizaje informado de testing con ejemplos). Esta propuesta permite incluir la educación en testing desde cursos introductorios de programación, ya que tiene como principales características que es:

- **temprana**: introduce el testing a los estudiantes desde el primer ejemplo de programas que ellos ven y que posteriormente usan cuando escriben sus ejercicios.
- **integrada**: permite incluir los conceptos de testing de manera continua y fluida como parte inherente de la programación, y no como conceptos aparte.
- **sutil**: utiliza métodos indirectos para enseñar a los estudiantes habilidades y conceptos de testing.

TILE es una propuesta que nace en el proyecto europeo QPeD (www.qped.eu), cuyo objetivo es mejorar la enseñanza en los cursos de programación, y está siendo desarrollado por investigadores y docentes de las universidades Open University, Philipps-Universität Marburg, Eindhoven University of Technology, Quarterfall, Universitat Oberta de Catalunya y Universitat Politècnica de València.

2. INNOVACIÓN DOCENTE

TILE ha sido inspirado en el aprendizaje dirigido por el testing (TDL por sus siglas en inglés - Test-Driven Learning). En [2], Janzen y Saiedian describen TDL como un enfoque pedagógico para enseñar programación, que implica la introducción y exploración de nuevos conceptos de programación utilizando ejemplos y ejercicios que evolucionan en torno al testing.

Los autores postulan que el principal beneficio de TDL es que permite agregar tempranamente los conceptos de testing sin la necesidad de cambiar el programa del curso. La idea que subyace en el TDL es que se beneficia de la forma en que se enseña/aprende programación, que mayoritariamente se realiza mediante el uso de ejemplos. Así, crear ejemplos relacionados a conceptos de testing tomarán el mismo esfuerzo que la creación de otros ejemplos típicamente creados para enseñar a programar.

Posteriormente, estos autores (Janzen et. al [3] y [4]) continuaron el trabajo hacia el desarrollo dirigido por las pruebas (TDD - por sus siglas del inglés Test Driven Development), que postula que se deben crear los casos de prueba antes de empezar a escribir alguna línea de código. Como consecuencia, hoy en día no hay una definición clara que permita a los docentes aplicar el aprendizaje dirigido por testing (TDL) en los cursos de programación. Por esta razón, para entregar guías claras, en este trabajo proponemos TILE: Aprendizaje Informado de Testing con Ejemplos.

Tradicionalmente, para enseñar algún concepto de programación:

1) El/la docente explica el concepto usando un ejemplo. Llamemos a este ejemplo *E1*. Como muestra, en [5], para comenzar a explicar el concepto de un objeto y sus métodos, el primer ejemplo que se usa está relacionado con Figuras (es decir, Círculos, Cuadrados, Triángulos, etc.).
2) El/la docente muestra ejemplos de ejecuciones para mostrar el resultado. Llamemos a estos ejemplos *E2*. Como muestra, en [5], las primeras instancias de la clase Circle se crean llamando al constructor y, posteriormente, se invocan métodos de esta instancia para ver qué sucede.
3) Los estudiantes practican con ejercicios que evolucionan sobre problemas de ejemplo, que llamaremos ejemplo *E3*, para los cuales deben aplicar lo que acaban de aprender.

La propuesta TILE plantea que los tres tipos de ejemplos mencionados anteriormente se pueden convertir en ejemplos que involucren conceptos de testing. Para explicar cómo se

puede hacer esto, se proponen tres tipos de TILEs: ejemplos de ejecución de pruebas, ejemplos de casos de prueba y ejemplos del dominio de pruebas.

3. PROYECTO

A continuación, se presentan los tres tipos de TILE que se proponen incluir en los cursos introductorios de programación.

3.1. TILEs de ejecución de pruebas

Cuando mostramos (o preguntamos) a los estudiantes qué sucede cuando ejecutamos un ejemplo de código de programación (*E2*), los docentes podemos, con un pequeño cambio de redacción, hacer un TILE con ese ejemplo. Para aclarar las ideas, veamos una muestra.

Considere el siguiente ejercicio introductorio en un curso de Python, donde se quiere enseñar a los estudiantes sobre variables, expresiones, `input` y `print`. Suponga que el/la docente entrega el siguiente programa sencillo a los estudiantes:

```
n = int(input("Enter a number: "))

square = n * n

print("The square is: ", square)
```

Posteriormente, el/la docente quiere que los estudiantes **ejecuten** este programa para ver qué sucede y aprender de él. Compare la redacción de las siguientes dos formas en que podríamos indicar esto a los estudiantes después de presentar el código de programación:

> *Ahora, cuando ejecutamos este programa, el usuario puede ingresar entradas a través del teclado y los resultados se mostrarán en la pantalla.*

VERSUS

> *Ahora podemos **probar** este programa ejecutándolo e ingresando datos de **entrada de prueba** a través del teclado y verificando **la salida resultante** en la pantalla.*

Como se puede observar, aplicar TILE a este ejemplo es sólo una nueva redacción del ejemplo que ya tenía el/la docente. Aunque es una diferencia muy sutil a la hora de decirles a los estudiantes lo que deben hacer, hay una gran ventaja al hacerlos conscientes de las pruebas e introducirlos en una fase muy temprana a la terminología de pruebas. En el ejemplo antes descrito se han presentado los conceptos de:

- probar el programa ejecutándolo,
- entrada de prueba (*test input*),
- comprobar la salida resultante (*expected result*).

Así, proponemos que todos los ejercicios y ejemplos que tratan de ejecuciones de programas estén redactados teniendo en cuenta conceptos de testing.

3.2. TILEs de casos de pruebas

Cuando el/la docente les entrega a los estudiantes ejercicios para que ellos resuelvan (es decir, ejemplos del tipo E_3), estos ejercicios deberían ir siempre presentados en forma de TILEs para asegurar que los estudiantes prueban su código. Esto puede ser tan simple como agregar una sentencia diciendo: *prueba tu código para verificar que funciona como pretendes*. Sin embargo, en las primeras fases de los cursos de programación, esto no siempre resulta efectivo, ya que los estudiantes simplemente ejecutan pruebas de *happy path* que corresponden a casos de prueba básicos sobre el comportamiento esperado del programa, en vez de casos de prueba que permitan detectar errores [6]; y, por lo tanto, no obtienen la conciencia necesaria sobre la importancia de las pruebas.

Una mejor manera de aplicar TILE a los ejercicios sería agregar ejemplos más concretos o ideas sobre posibles **casos de prueba** que el estudiante debería usar para verificar el funcionamiento de su código.

Estos casos de prueba de ejemplo no sólo deben incluir las pruebas del tipo *happy path*, sino también deben asegurar de que el código ha sido desafiado con algunos casos de borde y otras pruebas menos felices. Para ello, es necesario realizar un poco de esfuerzo y trabajo para aplicar TILE a los ejercicios de programación, ya que la descripción de ellos se hace un poco más larga. Sin embargo, existen numerosas ventajas:

- Los estudiantes comienzan a ver el valor de las pruebas porque encuentran errores en su código usando los casos de prueba dados cuando tal vez pensaban que habían terminado.
- Los estudiantes se acostumbran a ejecutar pruebas después de programar algo.
- Si en clases de programación más avanzadas no se le entregan casos de prueba de ejemplo a los estudiantes, ellos mismos van a realizarlas porque han reconocido el valor de ellas y ya no pueden prescindir de ellas.

Dependiendo del tipo de ejercicio, gusto personal, disponibilidad de oráculos, u otros factores externos a los contenidos del curso, se pueden crear ejercicios TILE de diversas maneras. A continuación, mencionamos algunas:

- agregando ejemplos de ejecuciones de prueba para que los estudiantes puedan verificar la salida de sus programas con esos ejemplos.
- agregando ejemplos de casos de prueba y aprovechando la oportunidad para familiarizar a los estudiantes con: la estructura de los casos de prueba (identificador, entradas, salidas esperadas), la definición de casos de prueba y necesidad de oráculos, casos de prueba abstractos, casos de prueba concretos, etc.
- obligar a los estudiantes a pensar en casos de prueba que expliquen las combinaciones necesarias y los valores límite que acepta su programa (si los prueban y descubren que su programa no funciona, comprenderán gradualmente el valor de las pruebas).
- Indicar oráculos paralelos a los estudiantes para que comprendan mejor el concepto de oráculos.

3.3. TILEs del dominio de pruebas

Como ha sido explicado previamente, aplicar TILE a los ejercicios de programación con casos de prueba y ejecución de pruebas, aunque requiere esfuerzo, es bastante sencillo.

Para el caso de los TILEs del dominio de pruebas (testing) es un poco diferente, pues requiere un poco más de creatividad y precaución.

Cuando un/una docente les explica los conceptos a los estudiantes y les muestra código de ejemplo, normalmente se inclina a utilizar un dominio conocido. Por ejemplo, si se quiere enseñar a los estudiantes las definiciones de clase y la herencia, a menudo se usan dominios como formas, animales, coches, personas, etc. Al definir una Forma como una clase padre y las clases hijas Círculo y Rectángulo, el/la docente ofrece un dominio a los estudiantes que ya pueden comprender, de modo que puedan aprender de mejor manera la jerarquía de las clases involucradas (en este caso las clases Forma, Círculo y Rectángulo), y tal vez incluso la diferencia entre Formas de manera abstracta, con los Círculos y Rectángulos que corresponden a formas concretas.

Si el dominio se ha elegido intencionalmente para permitir la transmisión de algunos conceptos, puede que no sea conveniente cambiarlo al dominio de pruebas. Sin embargo, si el dominio no influye en los conceptos que estamos enseñando, entonces los ejemplos se pueden reemplazar con ejemplos del dominio de pruebas directamente. Por ejemplo, si se quiere enseñar `input`, `print` y algunas operaciones básicas se puede utilizar el siguiente ejercicio con el dominio de pruebas.

Suponga que tiene un total de 15 casos de prueba de un programa Python. Por cada caso de prueba ejecutado, usted toma nota del número de errores que ha encontrado. Así, al finalizar las ejecuciones, usted tiene 15 números naturales: desde n1 a n15, que representan el total de errores que ha encontrado cada caso de prueba. Debe escribir un programa que pregunte por esos 15 números naturales como datos de entrada mediante el teclado y posteriormente determinar:

- *¿Cuántos casos de prueba no encontraron ningún error (es decir, encuentran 0 errores)?*
- *¿Cuántos casos de prueba encontraron entre 1 y 3 errores?*
- *¿Cuántos casos de prueba encontraron 4 o más errores?*

En la ejecución de ejemplo mostrada en la Figura 1 usted puede ver como su programa debe evitar que se introduzcan números negativos.

examples of test executions

```
>>> %Run
  Enter the number of bugs found by test 1: 3
  Enter the number of bugs found by test 2: 4
  Enter the number of bugs found by test 3: -5
  You cannot enter negative amounts.
  Enter the number of bugs found by test 3: 5
  Enter the number of bugs found by test 4: 6
  Enter the number of bugs found by test 5: 7
  Enter the number of bugs found by test 6: 0
  Enter the number of bugs found by test 7: 0
  Enter the number of bugs found by test 8: 1
  Enter the number of bugs found by test 9: 2
  Enter the number of bugs found by test 10: 6
  Enter the number of bugs found by test 11: 1
  Enter the number of bugs found by test 12: 2
  Enter the number of bugs found by test 13: 0
  Enter the number of bugs found by test 14: 0
  Enter the number of bugs found by test 15: 2
  Number of tests that have found 0 errors:  4
  Number of tests that have found between 1 and 3 errors:  6
  Number of tests that have found more than 4 errors:  5
```

Figura 1. Ejemplo de TILE de dominio

Como se puede observar, usar el dominio de pruebas en los ejercicios de programación que se entregan a los estudiantes requiere más creatividad, pero ayuda a los estudiantes a estar conscientes de cuáles son y cómo se relacionan los diferentes conceptos del testing.

4. CONCLUSIONES

Este trabajo presenta TILE, una propuesta de enseñanza temprana del testing a través de ejemplos. Nosotras creemos que TILE ayudará a resolver o permitirá disminuir los problemas ampliamente conocidos que ocurren cuando se integra la educación del testing en los cursos de programación.

En concreto, aplicar TILE a los ejercicios de los cursos de programación permitirá:

- Evitar el problema del huevo y la gallina respecto de las habilidades de testing y programación identificada en estos cursos donde se les ha pedido a los estudiantes que escribieran pruebas JUnit en una forma TDD (Test Driven Development) [7]. TILE no se trata de TDD en absoluto, ni comienza con pruebas unitarias automatizadas. TILE comienza diciéndoles a los estudiantes cómo deben probar los primeros programas que escriben.
- La actitud negativa de los estudiantes hacia las pruebas puede deberse a que todavía ven las pruebas como algo separado de la programación, algo que de repente debe hacerse y simplemente les da más trabajo. En TILE no presentaremos las pruebas como una actividad separada, se presenta y utiliza como una parte inherente de la programación (ya que es así).
- Respecto a los cursos de programación que dicen que los contenidos no se pueden extender más, es importante tener en cuenta que, si las pruebas se ven como un tema separado y adicional para enseñar, no se está enseñando programación de la

manera correcta. Si los docentes tenemos la idea de que agregar pruebas significa agregar más trabajo, o que las pruebas se pueden dejar de lado e intercambiar con otro tema, entonces estaremos transmitiendo el mismo mensaje a los estudiantes, contribuyendo a su actitud negativa.

- Con respecto a la carga de trabajo adicional, efectivamente la introducción de TILEs en los cursos y la realización de ejemplos y ejercicios que aplican TILE requerirá esfuerzo y aumentará la carga de trabajo una vez. Sin embargo, TILE vendrá con un repositorio de ejercicios reutilizable de código abierto, por lo que a lo largo del tiempo el esfuerzo será mínimo y los resultados valen la pena.

Como trabajo futuro planeamos realizar un experimento en las asignaturas introductorias de programación para evaluar empíricamente los beneficios de aplicar TILE a las guías y ejercicios creados para estos cursos.

REFERENCIAS

[1] Scatalon, L. P., Carver, J. C., Garcia, R. E., & Barbosa, E. F. (2019, February). Software testing in introductory programming courses: A systematic mapping study. In Proceedings of the 50th ACM Technical Symposium on Computer Science Education(pp. 421-427).

[2] Janzen, D. S., & Saiedian, H. (2006). Test-driven learning: intrinsic integration of testing into the CS/SE curriculum. Acm Sigcse Bulletin, 38(1), 254-258.

[3] Janzen, D., & Saiedian, H. (2008, March). Test-driven learning in early programming courses. In Proceedings of the 39th SIGCSE technical symposium on Computer science education (pp. 532-536).

[4] Desai, C., Janzen, D. S., & Clements, J. (2009). Implications of integrating test-driven development into CS1/CS2 curricula. ACM SIGCSE Bulletin, 41(1), 148-152.

[5] Barnes, D. J., Kölling, M., & Gosling, J. (2006). Objects First with Java: A practical introduction using BlueJ. London: Pearson Prentice Hall.

[6] Edwards, S. H., & Shams, Z. (2014, June). Do student programmers all tend to write the same software tests?. In Proceedings of the 2014 conference on Innovation & technology in computer science education (pp. 171-176).

[7] Gulati, S., & Sharma, R. (2017). Java Unit Testing with JUnit 5: Test Driven Development with JUnit 5. Apress.