



Stichting NIOC en de NIOC kennisbank

Stichting NIOC (www.nioc.nl) stelt zich conform zijn statuten tot doel: het realiseren van congressen over informatica onderwijs en voorts al hetgeen met een en ander rechtstreeks of zijdelings verband houdt of daartoe bevorderlijk kan zijn, alles in de ruimste zin des woords.

De stichting NIOC neemt de archivering van de resultaten van de congressen voor zijn rekening. De website www.nioc.nl ontsluit onder "Eerdere congressen" de gearchiveerde websites van eerdere congressen. De vele afzonderlijke congresbijdragen zijn opgenomen in een kennisbank die via dezelfde website onder "NIOC kennisbank" ontsloten wordt.

Op dit moment bevat de NIOC kennisbank alle bijdragen, incl. die van het laatste congres (NIOC2023, gehouden op donderdag 30 maart 2023 jl. en georganiseerd door NHL Stenden Hogeschool). Bij elkaar bijna 1500 bijdragen!

We roepen je op, na het lezen van het document dat door jou is gedownload, de auteur(s) feedback te geven. Dit kan door je te registreren als gebruiker van de NIOC kennisbank. Na registratie krijg je bericht hoe in te loggen op de NIOC kennisbank.

Het eerstvolgende NIOC vindt plaats op donderdag 27 maart 2025 in Zwolle en wordt dan georganiseerd door Hogeschool Windesheim. Kijk op www.nioc2025.nl voor meer informatie.

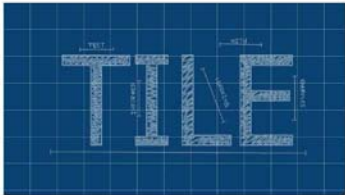
Wil je op de hoogte blijven van de ontwikkeling rond Stichting NIOC en de NIOC kennisbank, schrijf je dan in op de nieuwsbrief via

www.nioc.nl/nioc-kennisbank/aanmelden_nieuwsbrief

Reacties over de NIOC kennisbank en de inhoud daarvan kun je richten aan de beheerder:

R. Smedinga kennisbank@nioc.nl.

Vermeld bij reacties jouw naam en telefoonnummer voor nader contact.



Test Informed Learning with Examples (TILE)

Set the right example when teaching programming

NIOC 2023: Onstuimige vragen, creatieve oplossingen?

Niels Doorn

Open Universiteit, NHL Stenden

✉ niels.doorn@ou.nl, [nhlstenden.com](mailto:nielsdoorn.com)

🏠 research.nielsdoorn.nl

🆔 0000-0002-0680-4443

🌐 [nielsdoorn](https://nielsdoorn.nl)

🐙 [@niels76@mastodon.online](https://mastodon.online/@niels76)

We would love it if our IT graduates would become so called 'Test Obsessed'

Software testing is an important skill for software engineers

- **Little attention** is given to it
- Integrating software testing in **early** programming courses is **beneficial**
- There are **drawbacks** to integrating testing in programming courses

How can we improve teaching / learning strategies for software testing?

Test Informed Learning with Examples (TILE) is a way to integrate testing in education:

- **Early:** from the very first programming exercise on
- **Seamless:** a smooth and continuous way as an inherent part of programming
- **Subtle:** clever and indirect methods to teach testing

We developed four different types of TILES:

- Test **run** TILES
- Test **cases** TILES
- Test **message** TILES
- Test **domain** TILES

We can ask the students to **test** the program instead of asking them to **run** the program

Example of a test run TILE

Consider the following program:

```
n = int(input("Enter a number: "))
square = n * n
print("The square is: ", square)
```

Compare the wording of the following two ways:

1. Now let us **run** this program, the user can give input through the keyboard and the results will be shown on the screen
2. Now let us **test** this program by running it and **entering test input data** through the keyboard and **checking the resulting output** on the screen

Students often only test **happy path** execution

We can add **add more concrete examples of possible test cases** to create awareness of other useful test cases

TILE-ing exercises this way can be done in different ways

For example:

1. Adding **example test executions**
2. Adding **example test cases**
3. Forcing students to think about test cases explaining needed **combinations and boundary values**
4. Point the students to a **parallel oracle**

Example of a test case TILE: adding example runs

Consider this exercise:

➤ **Exercise:** *Implement a program that reads an integer corresponding to a month of the year and displays the name of the corresponding month. An error message should be displayed if the entered number does not belong to the range [1, 12].*

```
>>> %Run
Enter the number of the month: 5
May
>>> %Run
Enter the number of the month: 1
January
>>> %Run
Enter the number of the month: 13
Error: enter a number between 1 and 12
>>> %Run
Enter the number of the month: 0
Error: enter a number between 1 and 12
>>> %Run
Enter the number of the month: -3
Error: enter a number between 1 and 12
```

Example of a test case TILE: presenting test cases

➤ **Exercise:** *Implement a program that asks the user for a comparison operator: <, <=, >, >=, ==, != and 2 values. Your program has to display on screen the result (True or False) of the given operation applied to the two values.*

test id	test inputs			expected output
	operator	value1	value2	
1	<	12	4	False
2	>	100	40	True
3	==	"Hello!"	40	False
4	!=	100	"Python"	True
5	>=	98.67	0.45	True
6	<=	-100	40	True
7	<	24	"24K"	True
8	>=	"email"	"correo"	True

Example of a test case TILE: forcing students to think about test cases

➤ **Exercise:** *Write a program that receives as input an amount of euros, and displays as output the minimum breakdown in bills and coins for that amount on the screen. We assume that there are 500, 200, 100, 50, 20, 10 and 5 bills, and 2 and 1 coins.*

Then, students are asked to think about other tests they should run to ensure that the program has the desired behaviour

- Did they run a test with the amount 0 and with a negative amount?
- Did they make sure to choose different amounts in such a way that each bill and coin was returned at least once?
- Did they choose different amounts to test whether the program returns the minimum number of correct combinations of bills and coins?

Example of a test case TILE: forcing students to think about test cases

➤ **Exercise:** *Write a program that receives as input an amount of euros, and displays as output the minimum breakdown in bills and coins for that amount on the screen. We assume that there are 500, 200, 100, 50, 20, 10 and 5 bills, and 2 and 1 coins.*

For example:

- for 2, one coin of 2 must be returned (and not two of 1)
- for 10, one bill of 10 must be returned (and not two of 5)
- for 6, a bill of 5 and a coin of 1 must be returned (and not for example 3 coins of 2)
- for amount 12, a bill of 10 and a coin of 2 must be returned

TILES of this type hide **a subliminal message** about the importance of testing.

Example of a test message TILE

➤ **Exercise:**

Write a program that asks the user for something important and returns a billboard ASCII art.

```
>>> %Run  
Something important: Testing your code
```

```
          \|||||/  
          ( 0 0 )  
|-----oo0-----(-)-----|  
| Testing your code is important! |  
|-----0oo-----|  
          |_|_|_|  
          ||  ||  
          oo0  0oo
```

Example of a test message TILE

MadLibs is a word game where a player asks others for a list of words to substitute for blank spaces in a story, before reading the story aloud

MadLibs can be turned in all sorts of message TILES, e.g.:

➤ **Exercise:** Consider the following little MadLibs.

Whether you^(verb) computer programs to solve^(plural noun)
or just for^(noun) It is very important that you ALWAYS TEST
your code for^(adjective) bugs.

Example of a test message TILE

MadLibs can be turned in all sorts of message TILES, e.g.:

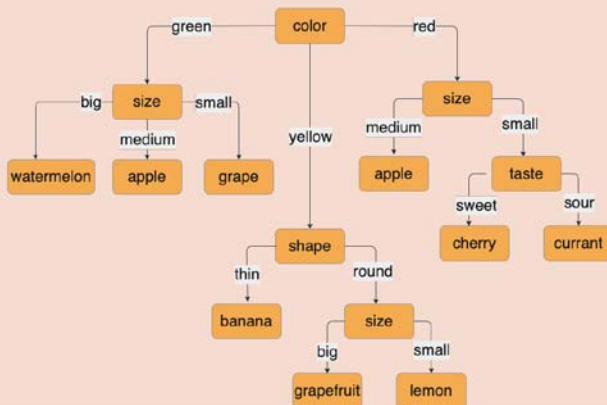
➤ **Exercise:** *Consider the following little MadLibs.*

We need to ask the following words:

- verb, for example: write
- plural noun, for example: problems
- noun, for example: fun
- adjective, for example: nasty

and use these words to fill in the placeholders in the figure. Try any combination of words, and when your program returns the result, read it out loud

➤ **Exercise:** Create a program that guides the user through the process of figuring out the type of fruit on hand. Use the following decision tree to build your program.



➤ **Exercise:** Create a program that guides the user through the process of figuring out the type of fruit on hand. Use the following decision tree to build your program.

```
>>> %Run
Color (green/yellow/red): green
Size (big/medium/small): big
watermelon

>>> %Run
Color (green/yellow/red): yellow
Shape (round/thin): round
Size (big/small): big
grapefruit
```

Example of a domain TILE

➤ **Exercise:** Imagine Ana wrote a program that sorts a list of numbers. Evidently, this needs to be tested. Create a program that guides her through the process of deciding whether she has tested sufficiently as shown below. Try to extend the program with possible cases that test other important things (at least two are useful to add).

```
>>> %Run
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

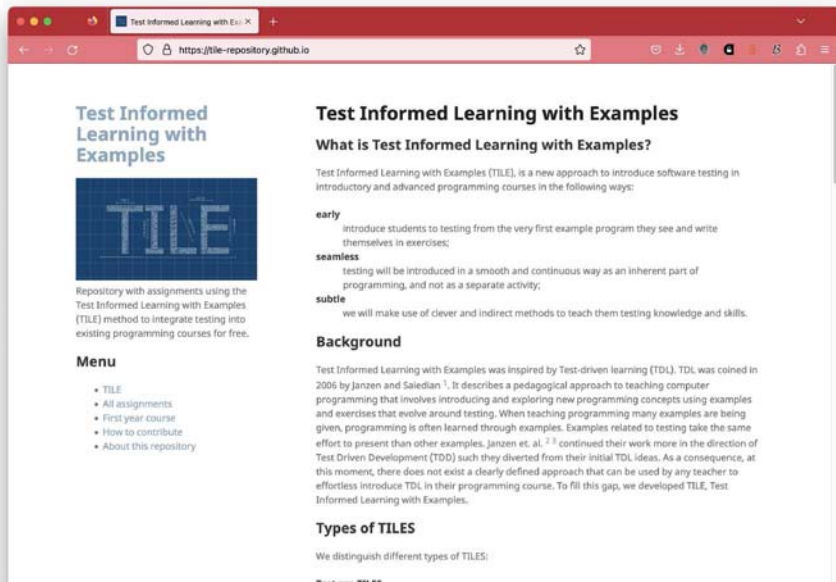
Well done!

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!
```

Repository with TILed exercises



The screenshot shows a web browser window with the address bar displaying `https://tile-repository.github.io`. The page content is as follows:

Test Informed Learning with Examples



Repository with assignments using the Test Informed Learning with Examples (TILE) method to integrate testing into existing programming courses for free.

Menu

- TILE
- All assignments
- First year course
- How to contribute
- About this repository

Test Informed Learning with Examples

What is Test Informed Learning with Examples?

Test Informed Learning with Examples (TILE), is a new approach to introduce software testing in introductory and advanced programming courses in the following ways:

- early**
introduce students to testing from the very first example program they see and write themselves in exercises;
- seamless**
testing will be introduced in a smooth and continuous way as an inherent part of programming, and not as a separate activity;
- subtle**
we will make use of clever and indirect methods to teach them testing knowledge and skills.




Background

Test Informed Learning with Examples was inspired by Test-driven learning (TDL). TDL was coined in 2006 by Janzen and Saiedian¹. It describes a pedagogical approach to teaching computer programming that involves introducing and exploring new programming concepts using examples and exercises that evolve around testing. When teaching programming many examples are being given, programming is often learned through examples. Examples related to testing take the same effort to present than other examples. Janzen et. al.^{2,3} continued their work more in the direction of Test Driven Development (TDD) such they diverted from their initial TDL ideas. As a consequence, at this moment, there does not exist a clearly defined approach that can be used by any teacher to effortlessly introduce TDL in their programming course. To fill this gap, we developed TILE, Test Informed Learning with Examples.

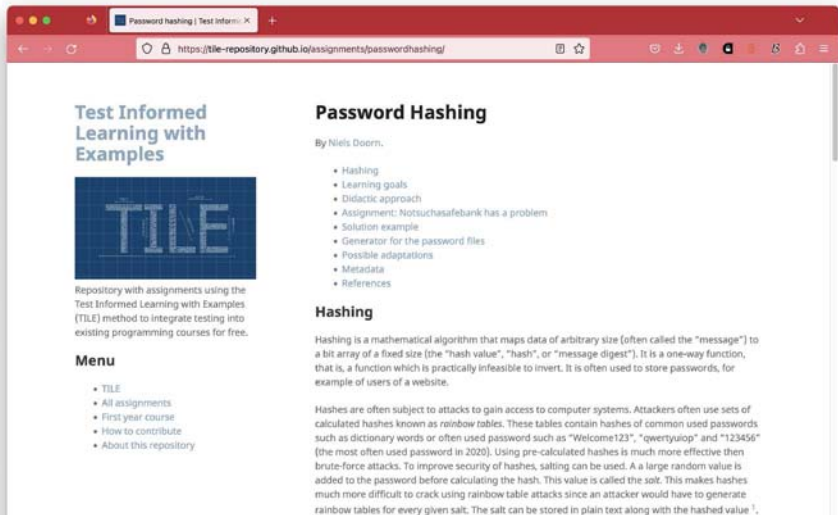
Types of TILES

We distinguish different types of TILES:

Repository with TILed exercises


- We created an open  GitHub repository
- Over 100 exercises (all types of TILs)
- Both  and  exercises (but easy to transform)
- Each exercise has meta data to make it easier to fit them in existing courses
- We encourage lecturers to contribute!

Repository with TILed exercises



The screenshot shows a web browser window with the address bar displaying `https://tile-repository.github.io/assignments/passwordhashing/`. The page content is as follows:

Test Informed Learning with Examples



Repository with assignments using the Test Informed Learning with Examples (TILE) method to integrate testing into existing programming courses for free.

Menu

- TILE
- All assignments
- First year course
- How to contribute
- About this repository

Password Hashing

By Nils Doorn.

- Hashing
- Learning goals
- Didactic approach
- Assignment: Notsuchasafebank has a problem
- Solution example
- Generator for the password files
- Possible adaptations
- Metadata
- References

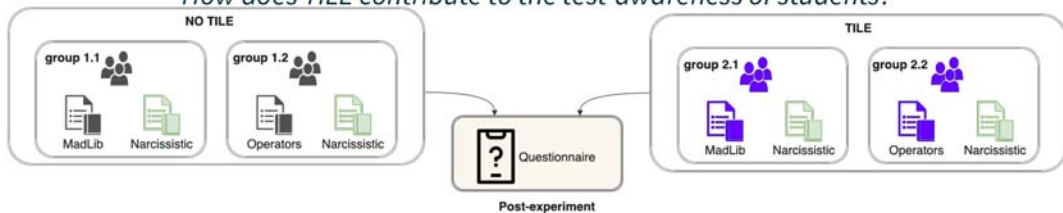
Hashing

Hashing is a mathematical algorithm that maps data of arbitrary size (often called the "message") to a bit array of a fixed size (the "hash value", "hash", or "message digest"). It is a one-way function, that is, a function which is practically infeasible to invert. It is often used to store passwords, for example of users of a website.

Hashes are often subject to attacks to gain access to computer systems. Attackers often use sets of calculated hashes known as *rainbow tables*. These tables contain hashes of common used passwords such as dictionary words or often used password such as "Welcome123", "qwertyuiop" and "123456" (the most often used password in 2020). Using pre-calculated hashes is much more effective than brute-force attacks. To improve security of hashes, salting can be used. A large random value is added to the password before calculating the hash. This value is called the *salt*. This makes hashes much more difficult to crack using rainbow table attacks since an attacker would have to generate rainbow tables for every given salt. The salt can be stored in plain text along with the hashed value ¹.

Initial exploratory study

How does TILE contribute to the test-awareness of students?



We measured:

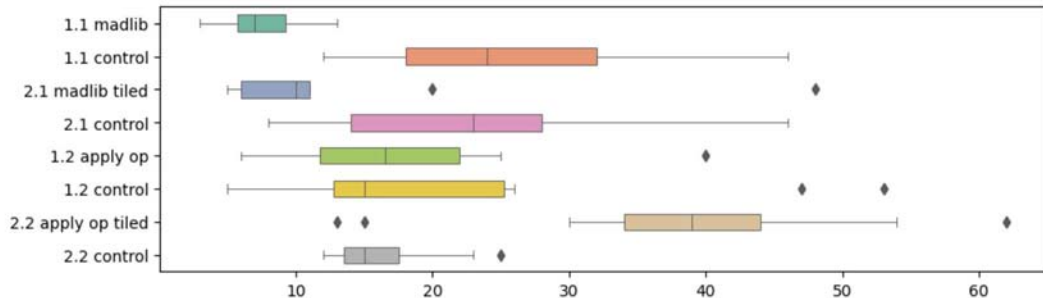
- the testing awareness of students, which is measured in number of testcases produced by students
- the time used to solve the exercises, measured in minutes
- the perception of the students about the confidence of their solution, measured in a 5-point Likert scale

Number of Testcases and Confidence of subjects ($n = 50$):

Groups	First exerc. #testcases	Control exerc. #testcases	Average Confidence	Std.dev.
non-TILEd	5	18	4,06	1,19
TILEd	33	23	4,12	1,03

Initial exploratory experiment

Time spent on the exercises:



Our next steps in TILE will be:

- Linking the Computer Science Educational Research theories to TILE
- Improve the repository by making the meta-data more accessible
- Adding exercises to the repository

Our next steps in Software Testing Education research will be:

- Getting a better understanding of the sensemaking of students and expert practitioners while designing test cases
- Develop instructional designs to improve software testing in education, including
- with the use of serious games, and study the effects in different educational contexts

Test Informed Learning with Examples:

- TILE as a new concept for test-aware introductory programming courses
- It allows **early, seamless** integration in a **subtle** way
- There are test **run**, test **case**, test **message** and test **domain** tiles
- Our initial study shows an increase of test cases created by students

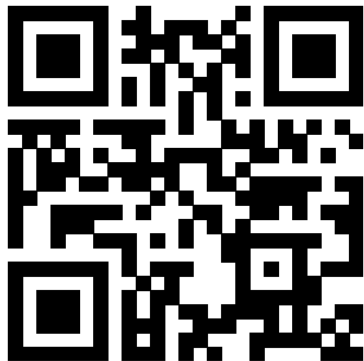
Thank you!

Read more about this research



edu.nl/utgcw

The repository can be found here



edu.nl/f9ptp

<https://research.nielsdoorn.nl>

Come join Martijn and me...

...at the Romanian Testing Conference

**Uncovering the Challenges
in Teaching Software Testing.**

Valuable talks, thoughts and insights
on how to enhance
software testing education.

Be there!

RTC
Romanian
Testing
Conference



Associate Lecturer
@ NHL Stenden
Hogeschool



Team Leader /
Lecturer Researcher
@ NHL Stenden Univ.
of Applied Sciences