How do students test software units, and how to improve this.

HBO-i Docentendag 2021, Utrecht, The Netherlands

Niels Doorn

October 8, 2021

1 Well, how did I get here?

Niels Doorn

Team leader, lecturer researcher @ NHL Stenden

PhD student @ Open Universiteit

niels.doorn@{nhlstenden.com, ou.nl}

Figure 1: HBO-i studiereis 2019

1.1 History of this research

- Talked about my master thesis on NIOC 2018
- Did the experiment on natural testing approach in 2019
- Wanted to present the prelimary results on NIOC 2020 (got cancelled due to covid)
- Continued with research on Procedural Guidance, Open Inf and TILE
- Formally started my PhD in 2021 with the QPED Erasmus+ project
- Presented our paper on ICSE-JSEET 2021

And now we are here!!

2 The importance of software testings

> Testing leads to failure, and failure leads to understanding.
>
> – Burt Rutan

2.1 Let's look at some failures triggered by COVID-19

While there are always software failures, during the pandemic the use of software changed leading to new failures:

- Two security researchers found a Zoom bug that can be abused to **steal Windows passwords**, another security researcher found two new bugs that can be used to **take over a Zoom user's Mac** [1].
- The badly thought-out use of Excel was the reason nearly **16000 corona virus cases went unreported** in the UK [2].
- The internal database used by the South Carolina Department of Health and Environmental Control (DHEC) to track COVID-19 cases experienced software problems that caused **incomplete case reporting** for several days. [3].

2.2 Finding defects earlier saves costs

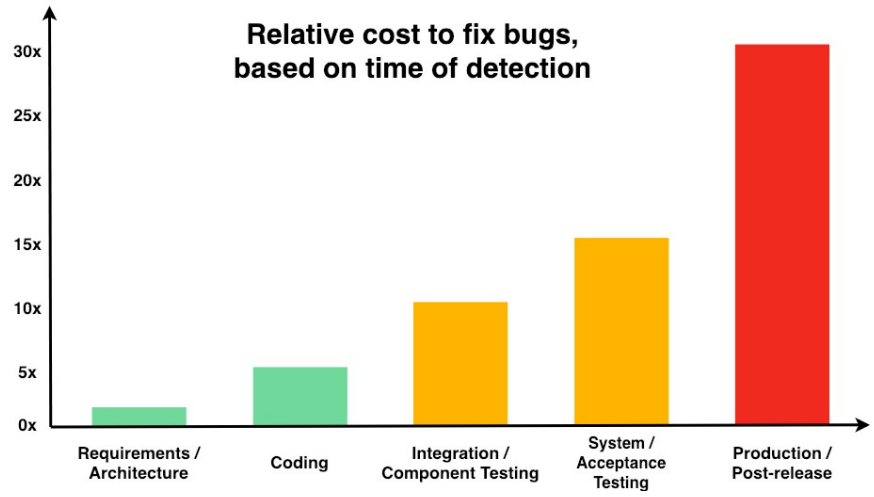

Figure 2: The cost of finding and fixing defects in different stages of development [4]

2.3 Everybody finds it important, yet, it is not tightly integrated into most CS programs (and in the industry engineers struggle with this too)

Educators agree that testing deserves more attention, and yet:

- It is not part of every CS program,
- and often it is an isolated part [5].
- The quality of test cases is low [6].
- Most students focus on happy-path testing [7].
- There is a gap between industry expectations and student skills [8].
- Students often perceive testing as not a very sexy subject.

There isn't much research *how* software testing can be taught effectively as an integral part of the program.

2.4 Getting our students to become 'Test Infected'

'Test Infected' is a term introduced by Erich Gamma and Kent Beck which describes the state of mind programmers have once they become so motivated that they are not satisfied with their programming until they have a complete and running test suite.

3 This presentation

- After this short intro,
- I want to present a study that we have done
- and look at the next steps of our research.

4 How do students test software units?

First we need to know what students do without any formal education on software testing.

4.1 Focus of this study

In this research, we investigated the ideas the participants have about testing *prior* to having formal education on this topic.

We aimed to study both their testing methods and the way they understand the concepts related to testing.

We included some of the questions from a study by Kolikant [9].

4.2 Research Setup

During this study we:

1. Did a related literature study.
2. Started with a technical report to report our findings.
3. Designed the experiment.
4. Did a test run with eight second year students to find any flaws.
5. Incorporated the needed changes in the setup.
6. Did a run with 31 first year students.
7. Gathered the results, analysed them.

We wrote everything down in a technical report.

4.2.1 Experiment

The experiment was as designed as follows:

- Pre-exercises inquiry.
- Exercises.
- Post-exercises inquiry.
- Interviews.
- Meta-analysis.

We also asked the participants to record the time they spent on the exercises and we encouraged them to provide us with feedback.

4.2.2 Exercise 3: smallest sequence of coins

```
/**
* Returns the smallest sequence of coins
* to represent the input argument.
* Possible coins:
* 1, 2, 5, 10, 20 and 50 cent and
* 1 and 2 euro (100 and 200 cent)
*/
public ArrayList<Integer> exchange(int amount) {
    ArrayList<Integer> result = new ArrayList<>();
```

```
    int[] coins = {200, 100, 50, 20, 10, 5, 2, 1};
    for (int coin : coins) {
        for (int i=0; i < amount / coin; i++) {
            result.add(coin);
        }
        amount = amount - (amount / coin) * coin;
    }
    return result;
}
```

4.2.3 Exercise 4: palindrome test

```
/**
* Input: A string
* Output: true if the string is a palindrome
*         otherwise false
*/
public boolean isPalindrome(String word) {
    // no body is provided
}
```

4.3 Results

4.3.1 Pre-exercises inquiry results

Some of the results of the pre-exercises inquiry:

- Many students place a lot of trust in the power of testing.
- Students favor happy path testing.
- Students indicate that it is best that end-users perform tests.
- Student believe testing in necessary, running it or just compiling it is no guarantee that a program works correctly.

4.3.2 Exercise results

Analyzing the students' answers, we have found four main categories of observations:

- Test approaches applied.
- Completeness of the test cases.
- Misconceptions.
- Programming knowledge.

4.3.3 The time spent on the exercises

This table shows the amount of time participants spent on the exercises.

Exercise	Average time spent	standard deviation
1	6:42	3:07
2	4:30	1:46
3	4:54	1:59
4	2:17	1:16

4.3.4 Correct / incorrect

We asked if the code was correct or not and to provide a test case if they believed the code was incorrect.

Exercise	Correct	Incorrect	Valid test case
1	11	11	7
2	4	21	19
3	10	13	n/a

4.3.5 Post-exercise inquiry results

Most important results of the post-exercises inquiry:

- Students believe that creating test cases helped to understand the code.
- Students believe that their test cases are sufficient.
- Students believe that they do test systematically.
- They also believe there is a change they have overlooked a test case.

4.3.6 Interview Results

We analysed the interviews to get a deeper understanding of the student's decisions and difficulties.

4.3.7 Test cases are based on code inspection

Code inspection is the basis for designing test cases:

> First, I've read the description of the exercise, after which I read the code thoroughly to determine its functionality. Otherwise, I am not able to determine the expected outcomes.

When there is no code, some students think of code:

> There was no code available, so I have to think about how it works. So I imagined how it could be implemented, to see how it should work.

4.3.8 When a bug is found, a test case for that bug is composed

Bugs are the main driver for test case design:

> Interviewer: And suddenly, you saw the error in the code?
>
> Student: Yes, and then I thought, I write [1,2,3] and then it is ready, on to the next one.

4.3.9 Application of wrong test strategies

Some students use random numbers as test cases:

> Student: For this, it is enough.
>
> Interviewer: You took some numbers, randomly, and looked ...
>
> Student: ... if they are correct. Yes.

4.3.10 Unnecessary or even impossible testing

The use of wrong data:

> Here I can add some characters and look how the program reacts because the program expects integers, but if I put in characters, then the program should chuck them out.

4.3.11 Lack of motivation

Some students show lack of motivation:

> I think that how important is the exercise ..., if it is for grading, then I should perform testing more elaborately then in cases of ordinary looking after the code. That is possible, nevertheless, then it works, but in cases of grading, then you should find all errors in the code.

4.3.12 Reading someone else code is difficult

One student told us:

> I experienced a lot of problems with the code conventions because I am used the place the brackets in a different way.

Another student mentioned:

> I did not understand the code really, because is naturally not my own code.

4.3.13 Pen and paper versus working on a computer

Missing the feedback from a computer is not helpful:

> It is difficult for me to do it just with pen and paper. It is easier to do it on a computer. Then, you can easily see what happens while running the code, what the code exactly does.

4.4 Conclusions

We found the following conclusions:

- Test cases are based on code inspection.
- When a bug is found, a test case for that bug is composed.
- There is a lack of systematic testing.
- Incomplete test sets.
- Wrong test strategies.
- Lack of motivation.
- Time spent to test is low.
- Unnecessary or even impossible testing.

The ICSE-JSEET paper can be found on the IEEE website.

5 How to improve software testing education

- What mental model do students use to create test cases?
- Development of Procedural Guidance for testing using the open informatics approach.
- Research into Test Informed Learning with Examples.

6 What mental model do students use to create test cases?

> We advocate that testing is model-based and exploratory, meaning that we can only make useful test models for test case design once we have obtained enough information about the testing problem which we can only get by first exploring, questioning, studying, observing and inferring.

We want to understand the *mental model* students use when designing test cases.

6.1 Experiment

Our main goal was to analyze the test models that they created with respect to a given problem in order to have initial insights of the mental model used by the students.

We used the TestCompass tool to create flow models and automate the design of test cases.

6.2 Assignment

A hardware store sells hammers (5 euros) and screwdrivers (10 euros). Over time however, their discount system has grown a bit complex. They have asked the nephew of the boss (who is studying computer science) to develop an application that can calculate the price a customer needs to pay when buying these products. They have the following discount rules:

- If the total price is more than 200 euros, then the client obtains a discount of 5% over the total;
- If the total price is more than 1000 euros, then the client obtains a discount of over 20% over the total;
- If the client buys more than 30 screwdrivers, then there is an additional discount of 10% over the total.

6.3 Example model

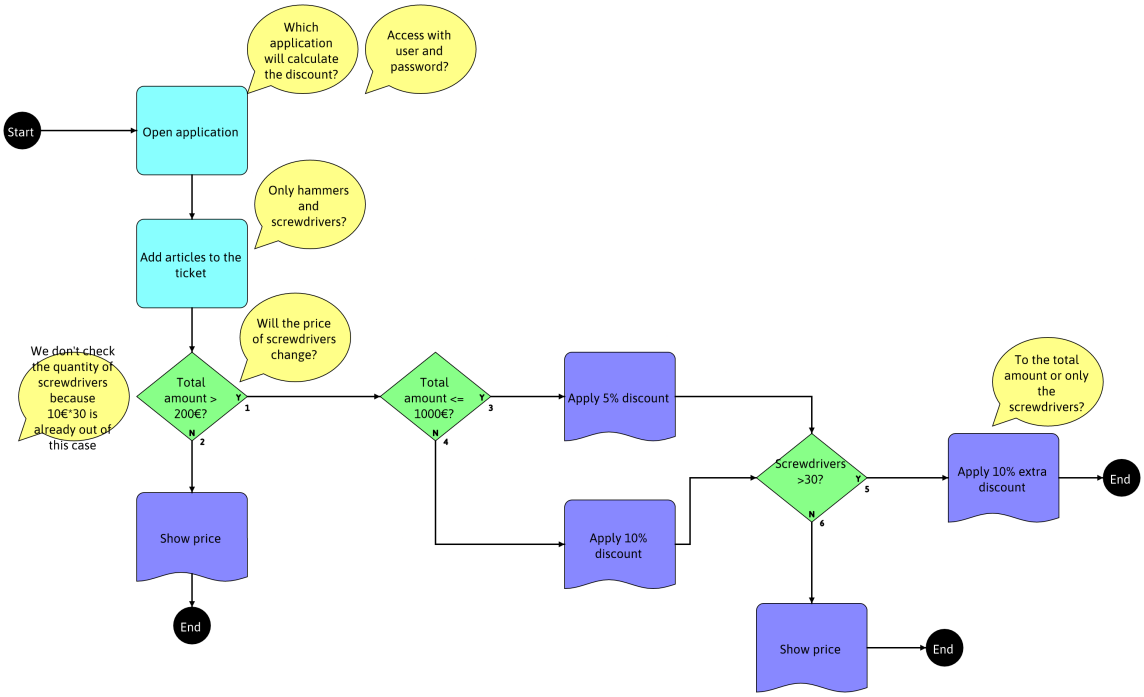

Figure 3: Example model

6.4 Results

- The models the students created were focused mostly on obvious functionalities and less on exploring the problem.

Students use a mixture of different approaches:

- Happy Path - students create the models focused only on the fulfillment of the requirements

- Development approach - students think about how they would implement the problem rather than test it
- Exploratory approach - elements are added to the models that are not mentioned directly in the requirements

6.5 Conclusions

- Indication that test case design is a complex problem requiring a different mental model than the creation of software
- We need to further investigate these differences
- Once we have a better understanding we can move towards studying an effective cognitive model of learning test case design

7 Procedural Guidance

> Even perfect program verification can only establish that a program meets its specification. The hardest part of the software task is arriving at a complete and consistent specification, and *much of the essence of building a program is in fact the debugging of the specification* [10].

7.1 How to get grip on the complexity.

- Developing quality software is a complex activity.
- Specifications are often incomplete or unclear.
- Many decisions have to be taken
- The order in which these decisions are taken can influence the complexity of the development process considerably.
- In practice, these decisions are often taken implicitly or even entirely not.

7.2 An educational device

- A stepwise development proces.
- It guides the student towards proficiency in the development method.
- When internalized, can be regarded as scaffolding and left behind.

7.3 Open Informatics

Open Informatics is a teaching method based on the same research results of Open Mathematics [11]:

The main pillars of Open Informatics are:

- Demonstrating positive messages about informatics and individual potential;
- Developing mathematical sense making by low-entry, high-ceiling problems and by great online resources;
- Establishing an encouraging atmosphere for effective teamwork.

This is based on the Growth Mindset idea of Carol Dweck [12].

7.4 Our approach

We want to use the open informatics approach with procedural guidance.

8 Test Informed Learning with Examples

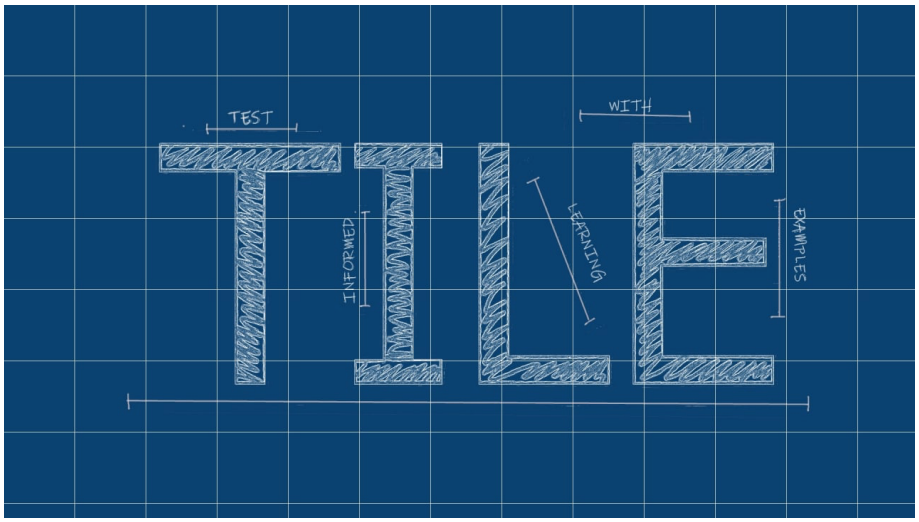

Figure 4: Test Informed Learning with Examples

8.1 Introduction

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.

8.2 Background

Test Informed Learning with Examples was inspired by Test-driven learning (TDL) [13]. 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.

- Programming is often learned through examples.
- Examples related to testing take the same effort to present than other examples.
- There is no clear approach on how to do this

8.3 Types of TILES

There are three main types of TILES:

- Test run TILES
- Test cases TILES
- Test domain TILES

8.3.1 Test run TILES

When we show (or asks) our students what happens when we run (or execute) a piece of programming code, we can, with a minor change of the wording make a TILE out of the example. For example, we can ask the students to *test* the program, or ask them to *assert* the right output instead of asking them to *run* the program.

8.3.2 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.

8.3.3 A *test run* TILE assignment (2)

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
```

8.3.4 A *test run* TILE assignment (3)

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?

8.3.5 Test cases TILES

Add concrete examples of possible *test cases* that the student should use to check the workings of their code. These example test cases should not just include the happy tests, but also make sure that their code is challenged on some corner cases and other less happy tests.

8.3.6 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.

8.3.7 A *test cases* TILE assignment (2)

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
```

8.3.8 A *test cases* TILE assignment (3)

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
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.

8.3.9 Test domain TILES

TILES of this type require a bit more creativity than the previous ones. In some programming exercises we use examples from a well known domain. For example, in explainations of Object Oriented Programming one might use the concept of shapes with classes such as `Rectangle` and `Circle` as a domain. If the domain does not influence the concepts we are teaching, then it can be replaced with examples from the testing domain directly.

8.3.10 A *test domain* 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.

8.3.11 A *test domain* TILE assignment (2)

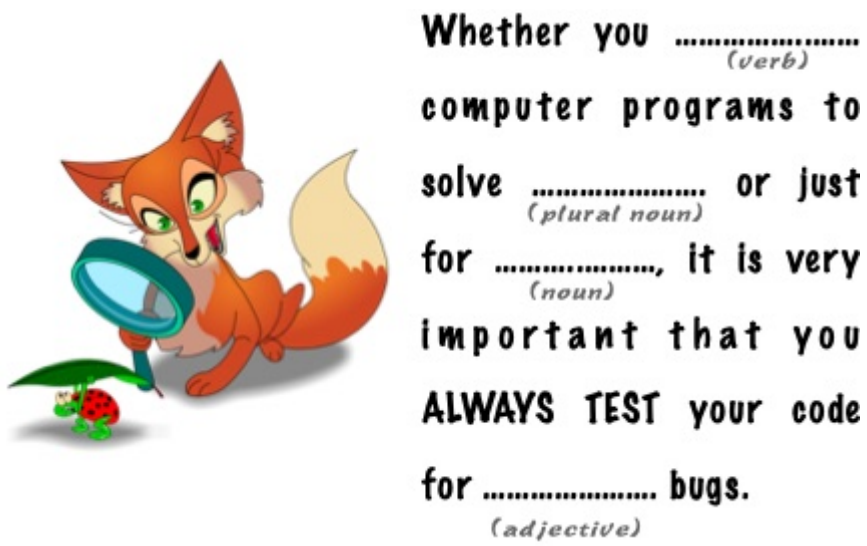

Figure 5: Example of a test domain TILE

8.3.12 A *test domain* TILE assignment (3)

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.

8.4 Our contributions so far

- We did a small experiment with bachelor students
- A group of lecturers are creating assignments for an open repository
- One complete course has been TILED and the results are very piositive

9 Summarize

To improving testing education we:

- studied the natural way of testing of students;
- did research into the mental model and cognitive processes of students;
- to design a didactical approach to start testing early
- using procedural guidance and an approach based on the growth mindset
- and we apply test informed learning with examples

10 Thank you!

Figure 6: Crowd debugging per excellence

11 And remember

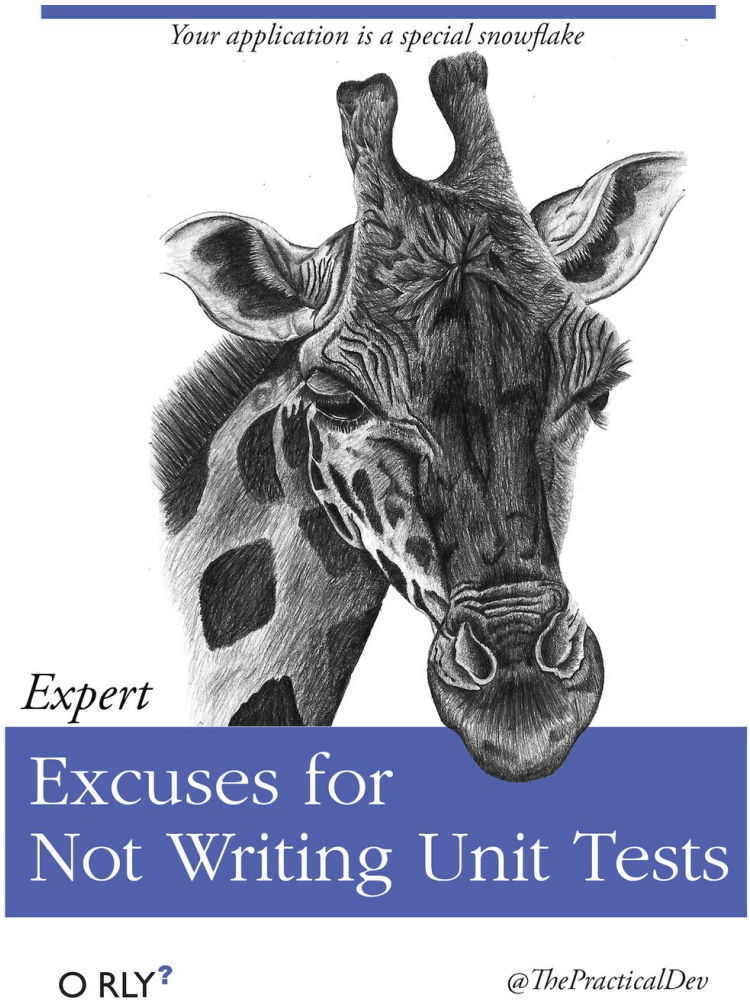

Figure 7: No excuses

References

[1] "Ex-NSA hacker drops new zero-day doom for zoom." https://tcrn.ch/2JuRTIT (accessed Feb. 27, 2021).

[2] "Excel: Why using microsofts tool caused covid-19 results to be lost - BBC news." https://www.bbc.com/news/technology-54423988 (accessed Feb. 27, 2021).

[3] "California COVID-19 vaccine rollout hit with software system problems." https://www.msn.com/en-us/news/us/california-covid-19-vaccine-rollout-hit-with-software-system-problems/ar-BB1cAgjn (accessed Feb. 27, 2021).

[4] Sanket, "The exponential cost of fixing bugs - DeepSource." https://deepsource.io/blog/exponential-cost-of-fixing-bugs/ (accessed Feb. 27, 2021).

[5] N. Doorn, "How can more students become 'test infected'," MSc thesis, Open Universiteit, 2018.

[6] S. Edwards and Z. Shams, "Do student programmers all tend to write the same software tests?" 2014, doi: 10.1145/2591708.2591757.

[7] L. M. Leventhal, B. E. Teasley, and D. S. Rohlman, "Analyses of factors related to positive test bias in software testing," *International Journal of Human-Computer Studies*, vol. 41, no. 5, pp. 717–749, 1994.

[8] A. Radermacher, G. Walia, and D. Knudson, "Investigating the skill gap between graduating students and industry expectations," in *Companion proceedings of the 36th international conference on software engineering*, 2014, pp. 291–300, doi: 10.1145/2591062.2591159.

[9] Y. Kolikant, "Students' alternative standards for correctness," Oct. 2005, pp. 37–43, doi: 10.1145/1089786.1089790.

[10] F. P. Brooks, "No silver bullet essence and accidents of software engineering," *Computer*, vol. 20, no. 4, pp. 10–19, Apr. 1987, doi: 10.1109/MC.1987.1663532.

[11] G. Alpár and M. van Hoeve, "Towards growth-mindset mathematics teaching in the netherlands," *Proceedings of Learning Innova*, vol. 2, pp. 1–17, Dec. 2019, [Online]. Available: https://www.narcis.nl/publication/RecordID/oai:research.ou.nl:publications%2F19e2743e-ca48-40d9-9dac-80be220456e4.

[12] C. Dweck, *Mindset-updated edition: Changing the way you think to fulfil your potential*. Hachette UK, 2017.

[13] D. Janzen and H. Saiedian, "Test-driven learning," in *Proceedings of the 37th SIGCSE technical symposium on computer science education*, Mar. 2006, pp. 254–258, doi: 10.1145/1121341.1121419.