Thematic Analysis – Baseline

Template

1. **Feelings on the development task**
    a) **Task1**
        - Easy / went smoothly **(1)**
        - Some difficulties / Needed more time **(2)**
    b) **Task 2**
        - No problem with the task (2)
        - Task was harder than last time and I struggled **(3)**
        - Task was harder than last time but no problem **(2)**
2. **Feelings on TDD to accomplish the task**
    a) **Task1**
        - Good experience **(1)**
        - Still not sure **(2)**
    b) **Task 2**
        - Good experience **(1)**
        - Mixed opinion **(1)**
        - Having troubles **(3)**
3. **Feelings on NO-TDD to accomplish the task**
    a) **Task1**
        - Good experience (I'm used to NO-TDD) **(2)**
        - Concerns on not testing enough some components **(1)**
        - **NO-TDD approach applied**
            - Wrote tests after the production code **(3)**
            - No tests **(2)**
    b) **Task 2**
        - Easier to use NO-TDD **(2)**
        - Not using TDD was harder **(2)**
        - **NO-TDD approach applied**
            - Wrote tests after the production code **(4)**
4. **Comparison between TDD and NO-TDD**
    a) TDD preference **(4)**
    b) NO-TDD preference **(3)**
    c) Depends on the task **(1)**

Task 1:

Student_01 (TDD):

- **Q1:** NO ANSWER
- **Q2:** NO ANSWER

Student_02 (TDD):

- **Q1**: I think TDD is very helpful, and I am glad that I can learn it
- **Q2**: I just needed more time and more exercise.

Student_03 (TDD):

- **Q1:** I honestly think it’s not as error prone as regular development. It increases software quality. And it assures that tests exist. But I’m honest, TDD is sometimes hard. Because you have to think different as usual methods.
- **Q2**: Task 4 - i really had to think deeply into this. Because implementing this breaks Task 3 tests

Student_04 (TDD):

- **Q1**: I have yet to fully understand how it works.
- **Q2**: The task I find it very useful to get a good understanding of the various steps to be done.

Student_05 (NO-TDD)

- **Q1**: I just programmed one task after the other, like we learned in the lessons ago.
- **Q2**: I think for me it’s easier to program a NO-TDD software, because I never did a TDD before. So, this is my "normal" way to code. Because we didn't need to write a test file it was quicker and less work.
- **Q3**: I liked that there were similarities with the previous tasks.

Student_06 (NO-TDD)

- **Q1**: I still have some difficult during the test phase, maybe is simpler doing it with TDD approach.
- **Q2**: For me is simpler because i can focus on the code.
- **Q3**: Are clear and not difficult to implement.

Student_07 (NO-TDD)

- **Q1**: Long story short, I’ve implemented all the function following the user story and, only after I’ve finished all of them, i started writing tests.
- **Q2**: NO-TDD may be faster when you are doing easy staff but following this kind of approach may incur in same difficult debugging session, there are the possibility to implement some code that may be never tested or not tested well.
- **Q3**: this development task has been useful to understand the difference with using TDD and its advantages in respect to use NO-TDD

Student_08 (NO-TDD)

- **Q1**: The knowledge from the further lectures helped me to accomplish the development.
- **Q2**: I’m not used to python, so the syntax was a bit complicated. But NO-TDD is classical programming and implementing of code, so i felt familiar with it.
- **Q3**: easy to understand, very well structured and in knew exactly what to do.

Student_09 (NO-TDD)

- **Q1**: NO ANSWER
- **Q2**: Thats nice activities but maybe we can discuss about that because that’s not really clear and there are some unknown places in my mind.
- **Q3**: These activities help and improves us.

Task 2:

Student_01 (NO-TDD)

- **Q1**: I have coded first and then wrote the test.
- **Q2**: It was ok.
- **Q3**: It was not so hard because based on the exercises we have done before.
- **Q4**: TDD is the best one to avoid any errors in the future.

Student_02 (NO-TDD)

- **Q1**: No certain approach
- **Q2**: It was hard not to use TDD.
- **Q3**: It was okay.
- **Q4**: I prefer using TDD.

Student_03 (NO-TDD)

- **Q1:** Plain write code, test later.
- **Q2:** Actually, with TDD i wouldn't have run into some pitfalls. I had to rewrite some code because i didn't know i broke some legacy ones. With TDD i would have known way earlier.
- **Q3:** It was a brain full task, but still really good. Wouldn't it be funny to run the code on an actual roomba?
- **Q4:** TDD: It reduces bugs, refactoring breaking changes are known like instantly, the tests are like documentation. Non-TDD: Bugs and functional errors are only known at the end testing state, really bad. I would now prefer TDD. But only if the functional features are clear. For discovering technology, i wouldn't use TDD for sure. But i guess it makes sense that you can't write good tests before the implementation for something like a quick throw away prototype.

Student_04 (NO-TDD)

- **Q1**: I implemented all the functions first and then wrote the various tests.
- **Q2**: I feel quite confident.
- **Q3**: I feel quite confident.
- **Q4**: With the NO-TDD approach, I seem to be able to better test the functions I implement.

Student_05 (TDD)

- **Q1**: During the programming lessons of TDD, I could only listen to the explanation or try to write down the code from the projector. Because for me it was impossible to do both at the same time, I tried to copy the code and so I don't know how to program well the TDD style. So today it was really hard for me.
- **Q2**: I think the Cleaning Robot was harder than the intelligent office, but maybe only because of TDD. Also, when you write TDD, you have to writ like two codes, the actual code and then the whole test code, so it takes more time.
- **Q3**: I think that I have a few mistakes in the non TDD code, but still I got everything, and the mistakes should be easy to fix. So, I prefer the non TDD programming, but maybe also because I'm used to it.

Student_06 (TDD)

- **Q1**: For me is more difficult to use this methodology. I prefer to create the code before testing it
- **Q2**: I had some difficult to understand the task, more complex.
- **Q3**: NO-TDD was simpler to apply, also the tasks were easier to implement. Anyway, i prefer a no-TDD approach.

Student_07 (TDD)

- **Q1**: At first TDD may seem trickier, but in a few exercises it became more and more easy to apply
- **Q2**: The exercise seems a little bit harder than the last one and it required me more time.
- **Q3**: Applying TDD force me to think in a way I'm not used to do, the idea to implement only the code necessary to pass the code it's something I'm not into yet, but i still prefer TDD in respect to NO-TDD.

Student_08 (TDD)

- **Q1**: It is hard to think the other way around as a software developer. But i think in my opinion it is very useful and if you are used to it, it really helps you a lot to improve your programming
- **Q2**: NO ANSWER
- **Q3**: As i said in a further answer the thinking is upside down and it's kind of learning a new logical thinking. I still prefer the no-TDD because i like it more to write the logic.

Student_09 (TDD)

- **Q1**: I think I didn't get it, but I tried so hard.
- **Q2**: If i am still alive probably I am developing my skills.
- **Q3**: NO ANSWER