

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Project scope</b>	<b>4</b>
2.1	Background . . . . .	4
2.2	Objectives . . . . .	4
2.3	Scope . . . . .	5
2.4	Constraints . . . . .	6
2.5	Assumptions . . . . .	6
2.6	Risks . . . . .	6
2.7	Deliverables . . . . .	6
<b>3</b>	<b>System plan</b>	<b>7</b>
3.1	Explanation of inputs . . . . .	7
3.1.1	Class lists . . . . .	7
3.1.2	Commentary on students . . . . .	7
3.1.3	Seating plan used in class . . . . .	8
3.1.4	Exam notes . . . . .	9
3.2	Explanation of outputs . . . . .	9
3.2.1	Suggested students for focus . . . . .	9
3.2.2	Suggested seating plans . . . . .	10
3.2.3	Average notes for the semester . . . . .	11
3.2.4	Individual student reports . . . . .	11
3.3	Implied structure of backend . . . . .	12
3.3.1	The list of students . . . . .	12
3.3.2	Raw comments . . . . .	14
3.3.3	Comment table . . . . .	16
3.3.4	Progress table . . . . .	18
3.3.5	Students for focus . . . . .	18
3.3.6	Report . . . . .	19
3.3.7	Spreadsheet of moyennes . . . . .	19
3.3.8	Exam results . . . . .	19
3.3.9	Databases . . . . .	20
3.3.10	Data privacy . . . . .	21
<b>4</b>	<b>Choice of programming language</b>	<b>21</b>
4.1	Previous personal experience . . . . .	21
4.2	New languages . . . . .	22
4.3	Python . . . . .	22

<b>5</b>	<b>Test driven development</b>	<b>23</b>
5.1	Building IT systems . . . . .	23
5.2	The waterfall method . . . . .	23
5.3	Reducing the distance between clients and developers . . . . .	23
5.4	TDD . . . . .	23
5.5	And next...? . . . . .	23
<b>6</b>	<b>The graphical user interface</b>	<b>23</b>
6.1	Choice of language . . . . .	23
6.2	Implementation style . . . . .	23
6.3	Views . . . . .	23
6.3.1	Control view . . . . .	24
6.3.2	Class view . . . . .	24
<b>7</b>	<b>A single executable file</b>	<b>25</b>
<b>8</b>	<b>Instruction booklet</b>	<b>26</b>
8.1	Page 1: introduction . . . . .	26
8.2	Page 2: control view . . . . .	26
8.3	Page 3: class view . . . . .	26
<b>9</b>	<b>Missing features</b>	<b>26</b>
<b>10</b>	<b>Conclusion</b>	<b>27</b>
<b>A</b>	<b>Original project proposal</b>	<b>28</b>
<b>B</b>	<b>Useful quotes</b>	<b>29</b>

# 1 Introduction

From the classic teaching literature, Bucheton and Soulé have described the act of teaching as a multi-agenda game of postures requiring good preparation and excellent micro-decisions [BS09]. In their model, they identify the crucial roles of the teacher in controlling the cadence, the atmosphere, the scaffolding and the relationships during the class, and how each supports the learning objective. Many of the teacher's most time-consuming tasks do not take place in the classroom: good preparation and strong follow-ups (auto-reflection, marking, parent-teacher interactions) work to support and complement the overall success of the student learning process.

For example, teacher responsibilities include the ability to [TODO: citation needed]

- plan and implement effective classroom management practices,

- design and implement effective strategies to develop independent learners,
- engage students in active, hands-on, creative problem-based learning,
- build students' ability to work collaboratively with others,
- maintain a safe, orderly environment conducive to learning,
- adapt instruction/support to students' differences in development, learning styles, strengths and needs, and
- write student reports to guide changes in instruction and practice, and to improve student learning.

Many of these tasks are "ripe for automation" [Swei15, p.? TODO], although I would also accept that some of these tasks should **not** be automated (even if they can be). As John Hattie explains in *Visible Learning* [Hat12], "*Expert teachers monitor learning and provide feedback.*" In my opinion writing student reports are a perfect example of a necessary evil: although time consuming (and potentially stressful) for the teacher, writing a student report forces the teacher to reflect on the progress of the student and at the same time manage the expectations of all partners - student, teacher, management and parent.

So which tasks should be automated? Why? And for whom? When I first started teaching, my natural character led to two bad teaching practices: I found it difficult to engage with the quieter, more reserved students; and I was so busy answering student questions that I left little time for taking notes. I felt my teaching (and so hopefully also my students' learning experience) would benefit from a tool which tracked my interactions in class in an attempt to shift the focus away from the "louder" students.

TODO: discussion on equality for students, and [Hat12] on answering too quickly.

But if I start recording a brief comment at an opportune moment after a positive (or negative) interaction with a set of students, I could also use this to build a reminder of the interactions per student: a useful capability when planning lessons, writing reports, and especially for parent evenings.

For the Master's thesis project undertaken for the GymInf formation, I chose to build a suite of tools to support a range of teacher tasks including capturing key interaction information, building individual student reports, suggesting teacher-student interactions for upcoming classes, organising seating plans, and creating spreadsheets of marks.

I will now explain the layout of this thesis. In sections 3 and ?? I outline the planned architecture of the system, explaining how it can be built incrementally. While I sometimes take the opportunity to explore alternative solutions, in general I mainly explain and defend my decisions.

The tools will be coded in Python. This choice of language is mainly because Python is the language we teach our students, see section 4, and I would like to take this opportunity to consolidate my Python skillset.

I also wanted to use this thesis to improve my coding style, exploring the industry technique of *test driven development* as explained in section 5. The majority of my contemporaries from university who ended up in university have highly recommended this coding style, and while it has disadvantages (as discussed later) there are strong reasons to having it as an option [Amman16]. TODO proper link to book.

TODO intro for chaps 7 and 8

Given the limited resources of this masters' thesis, not all of the desired functionality has been delivered. These missing features are described in section 9. Please note that the current codebase was designed with these features in mind, and that existing code should require minimal changes to incorporate these features as they are added.

## 2 Project scope

As Robinson points out in *BOSCARD: a scoping tool* [Rob19], for a project to be effective and efficient it is necessary (but alas not sufficient) to be clear about the project's aims from the start. BOSCARD is an acronym for background, objectives, scope, constraints, assumptions, risks, and deliverables. Although an industry standard, Robinson claims that the origins of this approach to project planning are unclear but likely originated with the consulting company Cap Gemini in the 1980s [Rob19, p. 181].

### 2.1 Background

*Provide background information that includes the reasons for creating the project and mentions the key stakeholders who will benefit from the project result.*

I am doing master's thesis in Computing and have chosen to create a suite of tools to aid teachers. These will be implemented through the *teacher assisting tools* system, called TAT. This will be aimed at assisting myself and other teachers in their daily work.

### 2.2 Objectives

*Describe the project goals and link each of them with related, SMART project objectives.*

There are common mistakes made by many new teachers, for example

- answering student questions immediately rather than leaving time for the class to think,
- allowing a subset of students to monopolise class interactions, and
- taking inadequate notes during class.

I will build the TAT system to help teachers to overcome these issues. As well as recording student interaction information, the system will also build individual student reports, suggest future teacher-student interactions for upcoming classes, suggest seating plans, and create spreadsheets of marks.

## 2.3 Scope

*Provide a high-level description of the features and functions that characterize the product, service, or result the project is meant to deliver.*

TAT will be able to:

- Actions during teaching.
  - Record, during class, an interaction with a set of students.
  - Suggest students for focus, see section 3.2.1.
  - Manually modify the seating plan as required.
- Actions outside of teaching.
  - Prepare student reports.
  - Suggest a seating plan. **Descoped.**
  - Review previous seating plans. **Descoped.**
  - Prepare an empty spreadsheet for marking and noting an exam.
  - Collate marks and calculate semester note.
- Actions to set up the class.
  - Add a class to the list of classes taught.
  - Configure the class lists.
  - Suggest a seating plan. **Descoped.**

The TAT backend will be linked to a GUI, run from a single executable file, with a local instance per teacher.

TAT will not connect to the internet. TAT will not cover data privacy concerns beyond what is currently used in standard teaching practices in Geneva, see section TODO ??.

All development and testing will be done on my personal machine, a MacBook Air, and be delivered to the examiner on this platform as he has the same operating system. Any work required to migrate to the school system will not be in the scope of this thesis.

## **2.4 Constraints**

*Identify the specific constraints or restrictions that limit or place conditions on the project, especially those associated with project scope.*

The thesis must be defended and marked by September 8<sup>th</sup>, 2023. Therefore the TAT system must be delivered to my thesis advisor before August 14<sup>th</sup> 2023. I am writing this thesis and the code for the TAT system alone: while a basic functionality is essential for all objectives, I anticipate future enhancements and even functional additions will occur.

The TAT system will eventually run on the school computers. TODO: find out what they are running. However, for this project all testing will be done on my local machine.

## **2.5 Assumptions**

*Specify all factors that are, for planning purposes, considered to be true. During the planning process, these assumptions will be validated.*

I will assume that classes have at most 24 students, and that the seating plan is arranged in the standard three-by-four blocks of pairs.

## **2.6 Risks**

*Outline the risks identified at the start of the project. Include a quick assessment of the significance of each risk and how to deal with them.*

Given the constraints, and especially the time limit, there is a large risk of some features being dropped from the first version. I want to first deliver basic functionality with a graphical user interface, and then add as many features as possible in the timeframe.

## **2.7 Deliverables**

*Define the key deliverables the project is required to produce to achieve the stated objectives.*

The TAT system will provide a basic GUI over a backend handling the tasks covered in the scope. In particular, the system should at least allow notes to be taken against student names and then suggest pertinent students for focus.

### 3 System plan

In his 2017 book "Clean Architecture" Robert Martin<sup>1</sup> is clear about why we invest time in the planning phase of an IT system:

*"The goal of software architecture is to minimize the human resources required to build and maintain the required system."* [Mart17, p. 5]

We can analyse a system by connecting its (physical or virtual) inputs and outputs. In this project, we have

#### Inputs

- Class lists
- Commentary on students
- Seating plan used in class
- Exam notes and weights

#### Outputs

- Suggested students for focus
- Suggested seating plans
- Average notes for the semester
- Individual student reports

The inputs clearly contain sensitive information, and the relevant laws and best practices with regards to student data will be discussed further in Section 3.3.10. It is also worth pointing out that the processed data (data during calculations and the outputs) is also sensitive. In general we should try to store as little personal data as possible. TODO citation needed.

### 3.1 Explanation of inputs

#### 3.1.1 Class lists

For each class of interest, we need to have a list of students who are members of this class. For each student, it is often useful to store a "given" name with which they like to be called in class. Other information is not essential (gender, age, etc.), and so we will not store it.

#### 3.1.2 Commentary on students

During the teaching process, the teacher makes useful judgements about students and groups of students. The teaching process obviously includes contact time during classes, but it can also include thoughts and decisions during the planning process, while marking homework or exams, or while evaluating a lesson *ex-post*.

Comments could take the form

- Excellent definition of Pythagoras' Theorem

---

<sup>1</sup>Robert Martin likes to be called "Uncle Bob".

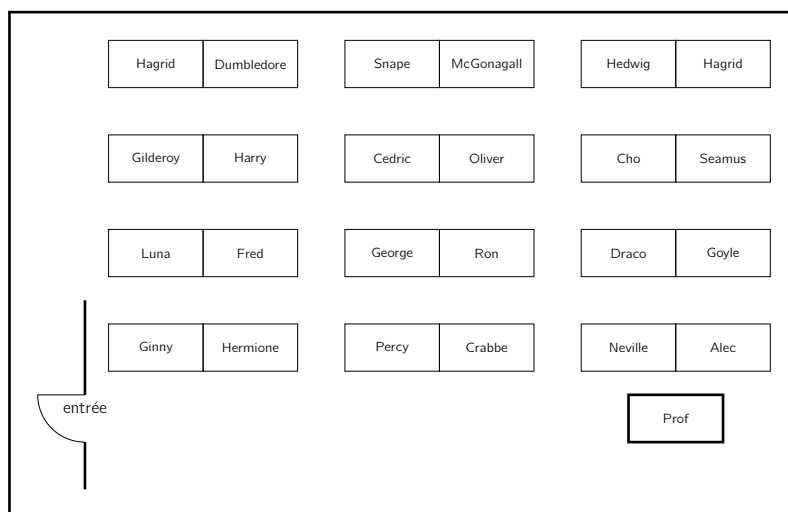


Figure 1: Example of a seating plan, created with the Tikz package of L<sup>A</sup>T<sub>E</sub>X

- Good explanation of photosynthesis on blackboard
- Very quick with past continuous exercises
- Chatting
- DNF (*homework not done*)
- Seemed unashamed that he did not know the formula for the area of a triangle.

To each comment should be associated the student, the class and the date. Thinking ahead, It would be very useful to also include whether or not the comment was positive or negative: see the "focus" and "report" outputs, respectively sections 3.2.1 and 3.2.4.

### 3.1.3 Seating plan used in class

Especially at the start of the year, but also just after holidays, it can be difficult to remember the given names of students. By asking the students to follow a seating plan *and then having this seating plan to hand*, the teacher has an easy way to refer to students by their given name. At the start of the year this also helps the teacher in learning the names. See figure 3.1.3 on page 8 for an example of a seating plan created using the tikz package of L<sup>A</sup>T<sub>E</sub>X. As far as possible, I followed the format of seating plans used during exams at Collège Rousseau : thus the seating plan layout should be familiar to both students and teachers.



### 3.1.4 Exam notes

The individual exam results will be needed in order to calculate the average note for each student for each semester, section 3.2.3. We will also need the weighting associated with each exam, and by including the date of the exam we can check the progression of students, section 3.2.4.

## 3.2 Explanation of outputs

### 3.2.1 Suggested students for focus

As a student teacher, I found it hard to balance my time equitably among the students in a class. Certain students, perhaps those who were louder or more confident, tended to capture my attention. By keeping a record of my interactions with students, it was possible to estimate an ordering of students by recent interactions, and so indicate students who require teacher *focus* during the next set of teacher-student interactions (probably during the next class).

This *focus* can be very simple: these are the students to whom the teacher addresses the opening questions. These opening questions can revise topics from the last lesson or prepare the students for the ideas to be tackled in this lesson. Students who respond positively to an early question are much more likely to volunteer answers later in the class TODO citation needed.

Let us model the interactions with a given student by simply counting the number of interactions on a given day.

Comments:  
01/06/2023 Harry gives conclusion of Pythag  
01/06/2023 Harry, Ron correct calc litt  
04/06/2023 Ron calculates angles in triangle  
08/06/2023 Harry calculates hypotenuse

We define a function  $f_H : \text{Dates} \rightarrow \mathbb{N}$  to count the interactions with Harry

$$f_H(d) = \begin{cases} 2 & \text{if } d = 01/06/2023 \\ 1 & \text{if } d = 08/06/2023 \\ 0 & \text{otherwise.} \end{cases}$$

and a corresponding function  $f_R$  for the interactions with Ron

$$f_R(d) = \begin{cases} 1 & \text{if } d = 01/06/2023 \\ 1 & \text{if } d = 04/06/2023 \\ 0 & \text{otherwise.} \end{cases}$$

On a given date  $t$  we would like to create a weight operator,  $W_t$ , which would map these functions to a real number. This operator should measure the number

of recent interactions, giving more weight if there have been more interactions. For example, Harry has had more interactions than Ron and so we expect  $W_t(f_H) > W_t(f_R)$ .

Consider calculating the students for focus just before the holidays or just after the holidays. Assuming the list of comments has not changed, then ideally the **ordering** of the students by weight would not change. Thus the students for focus should not be dependant on when the algorithm was run. A straightforward property with this behaviour is that all weights change by the same constant. In probability, this is called the *memoryless* property [Norr98] and implies an exponential weighting of the counts:

$$W_t(f) = \sum_{d \in \mathcal{D}} f(d) e^{k(d-t)}$$

where we sum over all dates<sup>2</sup> and  $k > 0$  is a constant. Then recalculating the weights after  $n$  days gives

$$W_{t+n}(f) = \sum_{d \in \mathcal{D}} f(d) e^{k(d-(t+n))} = e^{-kn} \cdot \sum_{d \in \mathcal{D}} f(d) e^{k(d-t)} = e^{-kn} \cdot W_t(f),$$

so all weights are rescaled by  $e^{-kn}$  which does not depend on  $f$ .

To pick the constant  $k$ , consider the desired relative weighting between a student to whom we made a comment yesterday, and another who received two comments quite a while ago (but two comments on the same day). How much time should pass in order that both students receive the same weight? Completely subjectively, I assume one week should pass, which gives

$$1e^{0k} = 2e^{-7k} \Rightarrow k = \frac{1}{7} \ln 2 \approx 0.1.$$

Ideally negative interactions should **not** be counted in this weighting; otherwise "difficult" students would be unlikely to ever be chosen for positive interactions.

### 3.2.2 Suggested seating plans

Most classes have 24 students arranged in 12 pairs, with the 12 pairs at desks arranged in 3 columns by 4 rows, see figure 3.1.3 on page 8. There are two obvious seating arrangements : using the ordering from the class list<sup>3</sup> or using a random seating plan. For testing and repeatability it is much easier to use a deterministic algorithm, and so I have preferred using the class list as a default seating plan.

As the year progresses we have more information about the students: pairs that work well together, students who work at similar speeds, students which have different strengths. Could we use the student comments and the exam results to suggest seating plans suited to particular lessons?

<sup>2</sup>Clearly, as the number of comments is finite this is really a finite sum.

<sup>3</sup>The class list is alphabetic on surname. However, due to data privacy concerns the system does not have access to the surname: it only has the email username, see section 3.3.10. The alphabetic ordering of the usernames is not necessarily the same as the alphabetic ordering of the fullnames. To avoid this problem I use the class list ordering directly rather than the usernames.

### 3.2.3 Average notes for the semester

The notes in Genevan institutions range between 1.5 and 6.0. The average note for the semester is a weighted sum of the individual notes of the marked exams of the semester. The individual notes are rounded to the nearest half, whereas the *semestriel* notes are rounded to the nearest tenth. Mathematically,

$$n_s = R_{0.1} \left( \frac{\sum_{i \in \mathcal{I}_s} w_i \cdot n_i}{\sum_{i \in \mathcal{I}_s} w_i} \right)$$

where the index runs over the marked exams of the semester, and  $R_s$  is the function that rounds to the nearest multiple of  $s$ .

It is worth pointing out that the rounding function is *not* continuous. While this is obvious, it can make the semestriel note surprisingly sensitive to small changes in individual exam notes. This also true for the end of year note, which is the **rounded** average of the semestriel notes:

$$n_y = R_{0.1} \left( \frac{n_{s_1} + n_{s_2}}{2} \right).$$

In general the pass mark is 4.0, and the rounding functions are a big advantage for weak students. Consider a concrete example, where before rounding a student receives 3.85 for the first semester and 3.95 for the second. Then the final end of year note is

$$n_y = R_{0.1} \left( \frac{R_{0.1}(3.85) + R_{0.1}(3.95)}{2} \right) = R_{0.1} \left( \frac{3.9 + 4.0}{2} \right) = R_{0.1}(3.95) = 4.0$$

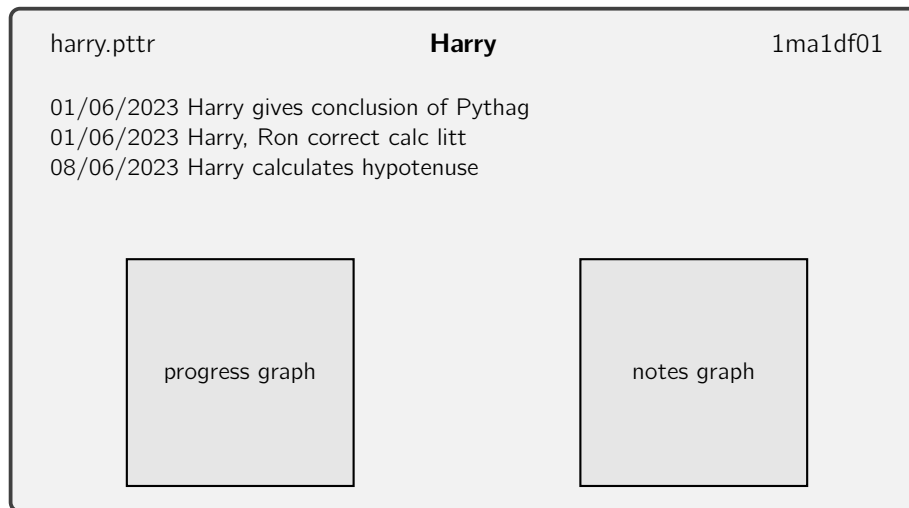
and the student passes! Fingers crossed this also applies to Master's theses.

I would like the system to calculate the semestriel and end of year notes for each student.

### 3.2.4 Individual student reports

It is often useful to have a quick overview of an individual student's progress. Perhaps beforehand, while planning which students will tackle which activities, or afterwards, when analysing the efficacy of a sequence of lessons. More concretely, such an overview would be very useful when meeting parents, when giving notes, and when asked to comment on their progress in the *conseils de classes* at the end of each semester.

Information that would be useful would be the student name, their given name, the class, a list of the comments concerning this student, their exam notes over time, and a graphical visualisation of their progress. Roughly, the overview would include an A4 page per student which looks something like this:



### 3.3 Implied structure of backend

By "backend" I mean the data access layer of the system. By connecting the inputs and outputs we can assess what intermediate processing will be required, and what functionality can be shared. The schematic in figure 3.3, on page 13, is a *Data Flow Diagram* for the TAT system. I have used the Yourdon-Coad notation where the processes (functions) are circles and data is represented by rectangles. Flow of control (transfer of information) is represented by arrows, see [Coad91].

The data flow diagram in figure 3.3, on page 13, is a very high-level description of a possible system. To make concrete decisions about how to implement the TAT system it will help to define the expected inputs and outputs. In this section I will outline a basic backend based on files. An implementation based on files is simpler and quicker, and so more suited for a master's project, but in the future I would expect the backend to handle updates via ACID database queries. I will briefly explain the advantages of databases over files in subsection 3.3.9. I will finish this section by touching on the data privacy laws that apply in Geneva; in particular the system will use only a minimum amount of personal data.

#### 3.3.1 The list of students

The **list of students** is a list of students in each class. I will use a text file for each class, storing a list of user-ids for each student. For example, for the class 1in1dfb01 I would create a textfile called "1in1dfb01.txt":

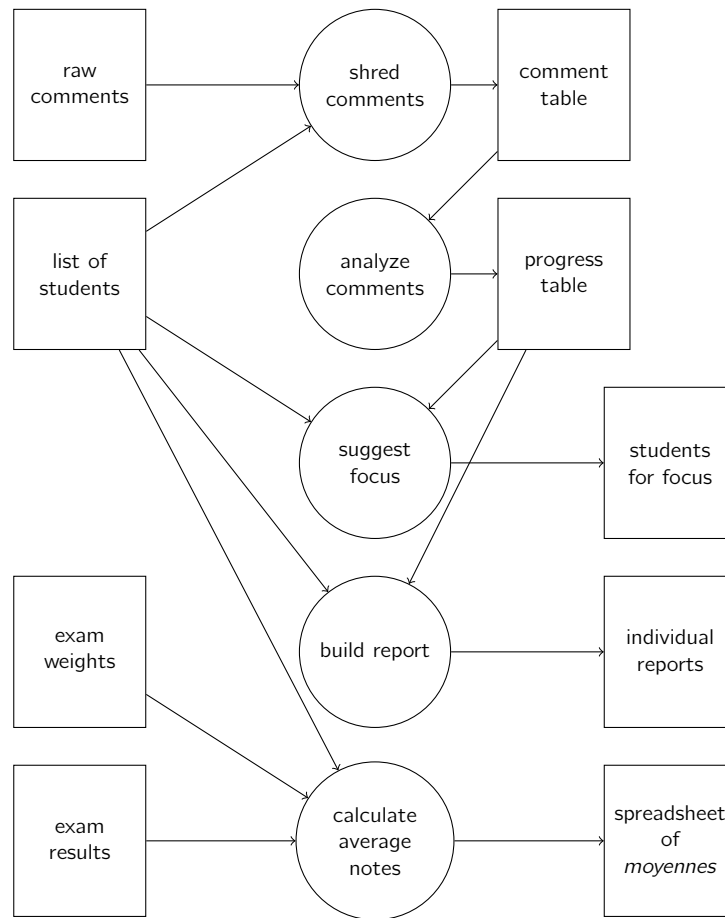


Figure 2: Data flow diagram for the backend of the TAT system.

```
1in1dfb01.txt
```

```
harry.pttr
ronald.wsly
marie.cr
richard.fynmn
```

This also seems a pertinent place to store the given name of each student. Currently the user-id starts with the first name of the student followed by a fullstop followed by (roughly) the consonants of the surname,

```
user_id = firstname + "." + consonants(surname).
```

A regular expression of the form `^[a-z\-]+` strips the student's first name from

their user-id. In Python, we use the `re` package, for example

```
firstname = re.search(r"^[a-z\-]+", user_id).group(0).capitalize()
```

Applying this function to the students above would yield

```
["Harry", "Ronald", "Marie", "Richard"]
```

However, there are two issues with assuming that this function will always yield the given name of a student. Firstly, we do not control how the user-ids are created, and so we cannot guarantee that this function will work in the future. Secondly, a student might want to be called by a different name (for example their second name).

I decided to add an optional comma-separated second item which designates the given name of a student in a class: this will be the name shown on the seating plan. For example, if Ronald Weasley wished to be called "Ron" and Richard Feynman preferred "Dick", then the file for the class list would be

```
1in1dfb01.txt  
  
harry.pttr  
ronald.wsly, Ron  
marie.cr  
richard.fynmn, Dick
```

### 3.3.2 Raw comments

The **raw comments** was originally stored in a text file designed to be read and updated by the teacher. For this project, I have used the same design. A comment is written on a new line in the file, prefaced with a "+" or "-" depending on whether the comment is positive or negative. Then I include a list of students, followed by a comment.<sup>4</sup>

Examples of positive comments:

```
+Harry gives conclusion of Pythag  
+Harry, Ron correct calc litt for  $(a+b)^2$ 
```

Examples of negative comments:

```
-Harry moaning about scar  
-Harry, Ron chatting
```

Before lines of comments there must be a line with the name of the class and, independently, the date. So a valid state for the comments file might be something like the following

---

<sup>4</sup>The text in the comment can handle  $\text{\LaTeX}$  commands

comments\_file\_v0.1.txt

```
01Apr2023
1indfb01
+Harry gives conclusion of Pythag
+Harry, Ron correct calc litt for  $(a+b)^2$ 

2indfb01
+George TN for Dijkstra

02Apr2023
1indfb01
-Ron DNF
+Harry, Ron TN for Pythagore
```

(Note that the vertical spaces are optional and aid human readability.)

Given that this is a first implementation, but that nevertheless we want an efficient backend, we can improve the parsing of this file by demanding that the first character in each line defines the information that this line will contain:

- "+" or "-" will be followed by a comma separated list of students and a comment.
- "d" will be followed by a date in "ddMMMyyyy" format.
- "c" will be followed by a class name.

comments\_file\_v1.1.txt

```
d01Apr2023
c1indfb01
+Harry gives conclusion of Pythag
+Harry, Ron correct calc litt for  $(a+b)^2$ 

c2indfb01
+George TN for Dijkstra

d02Apr2023
c1indfb01
-Ron DNF
+Harry, Ron TN for Pythagore
```

### 3.3.3 Comment table

The **comment table** is a pandas data frame<sup>5</sup> created from the raw comments. The *shred comments* functions read the raw comments from top to bottom, scraping the information in a comment along with the associated student, date, the course name, and the sentiment (whether the comment is positive or negative). A comment in the raw comments file can concern multiple students, so if necessary a line in the data frame is duplicated for each individual student. An example is given on page 17, restricting to the columns "Student", "Date", "Course", "Info" and "Sentiment".

NB: I had hoped that the TAT system would also keep track of missed homeworks (*devoirs non-fait* in French, hence "DNF" in the comments). I have decided to punish students with a electronic *renvoi* for every second homework missed. Of course, it is quite a lot of work to track which students have missed a second homework. The TAT system can easily count the DNFs through time which eliminates this bureaucracy (it also avoids mistakes and so is fairer.) The counting of DNFs is implemented in the backend but not yet wired up in the GUI, and so I do not mention it elsewhere in this thesis.

---

<sup>5</sup><https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html>



comments\_file\_v1.1.txt

d01Apr2023  
 c1indfb01  
 +Harry gives conclusion of Pythag  
 +Harry, Ron correct calc litt for  $(a+b)^2$

c2indfb01  
 +George TN for Djikstra

d02Apr2023  
 c1indfb01  
 -Ron DNF  
 +Harry, Ron TN for Pythagore

comments\_table\_DataFrame  $\subset$  progress\_table\_DataFrame

Student	Date	Course	Info	Sentiment	Weight	Progress
harry.pttr	01Apr2023	1indfb01	+Harry gives conclusion of Pythag	1	0.9	1
harry.pttr	01Apr2023	1indfb01	+Harry, Ron correct calc litt for $(a + b)^2$	1	0.9	2
ronald.wsl	01Apr2023	1indfb01	+Harry, Ron correct calc litt for $(a + b)^2$	1	0.9	1
george.wsl	01Apr2023	2indfb01	+George TN for Djikstra	1	0.9	1
ronald.wsl	02Apr2023	1indfb01	-Ron DNF	-1	0.0	0
harry.pttr	02Apr2023	1indfb01	+Harry, Ron TN for Pythagore	1	1.0	3
ronald.wsl	02Apr2023	1indfb01	+Harry, Ron TN for Pythagore	1	1.0	1

### 3.3.4 Progress table

In the **comments table** the columns were "Student", "Date", "Course", "Info" and "Sentiment". To these columns we adjoin the "Weight" and "Progress" columns to create **progress table**, which (like the comments table) is also implemented as a pandas DataFrame. See the example on page 17.

The *weight* of a positive comment is a memoryless function of the time duration between now and the date associated with the comment, see subsection 3.2.1 on page 9:

$$\text{weight} = e^{-kd}$$

where  $d$  is the age of the comment in calendar days and  $k = 0.1$  is a constant. For example, the weight of a comment made today is  $e^{-0.1 \cdot 0} = 1$  while the weight of a comment made yesterday is  $e^{-0.1 \cdot 1} \approx 0.9$ . Comments with a negative sentiment have a weight of 0 (forgive and forget).

The *progress* is the running cumulative sum of sentiments filtered on that student in that class. If we filter the progress table on `harry.pttr` in course `1indfb01` then we see

Date	Info	Sentiment	Weight	Progress
01Apr2023	+Harry gives conclusion of Pythag	1	0.9	1
01Apr2023	+Harry, Ron correct calc litt for $(a+b)^2$	1	0.9	2
02Apr2023	+Harry, Ron TN for Pythagore	1	1.0	3

### 3.3.5 Students for focus

We sort the set of students in a given course to return a list of **students for focus**, a list ordered in increasing need for teacher-student interactions. This "need" is approximated by a function of the distribution of the comments over time, as explained in subsection 3.2.1 on page 9. In practice, it is simply the sum of the weights in the progress table. Referring to the example on page 17, the sum of weights for the students in class `1indfb01` is

$$\{\text{"harry.pttr": 2.8, "ronald.wsl": 1.9}\}.$$

So as a list ordered on increasing weight, the students for focus would be

$$[\text{"ronald.wsl"}, \text{"harry.pttr"}].$$

Of course, other students in this course have a default weight of zero if there are currently no comments associated to them. So if the course also included `"hermione.grngr"` then the ordered list of students for focus would be

$$[\text{"hermione.grngr"}, \text{"ronald.wsl"}, \text{"harry.pttr"}].$$

### 3.3.6 Report

The **report** is a pdf document containing an A4 page per student. On each page we have the student-id, the student's given name, the course, and a list of comments associated with this student. The TAT system first creates a text-based .tex document, and then calls a system function which executes pdflatex<sup>6</sup> to typeset the report as a .pdf document.

### 3.3.7 Spreadsheet of moyennes

The TAT system can shred spreadsheets of individual exams to create a table with the exam name, exam date, course, student-ids and associated notes. It then creates a **spreadsheet of moyennes** with a worksheet per course and a table of notes with the exams as columns and the student-ids as rows.

So far, the description above explains how a spreadsheet of static data is created (albeit static data collated in a useful single file). The TAT system goes further, automatically creating a table of cells containing formulae (rather than static data) which calculate the student *moyennes* for the provisional notes (NIPs), the first semester notes (S1), the second semester notes (S2) and the implied end of year note (EOY). This allows the teacher to fill-in or correct individual exams and see the effect on the notes to be officially declared. Further, by leaving rows for the individual exam weights the teacher has the flexibility to specify which exams are noted and which are formative, which exams count double, as well as choosing the weight of the semestriel exams.

The algebra for the calculation of the notes was given in subsection 3.2.3, on page 11. The equivalent spreadsheet formulae are

- "RAWNOTE=SUMPRODUCT(EXAMWEIGHTS, EXAMNOTES)/SUM(EXAMWEIGHTS),
- "NOTE\_Sx=ROUND(RAWNOTE\*10)/10", and
- "NOTE\_EOY=ROUND(AVERAGE(NOTE\_S1;NOTE\_S2)\*10)/10".

I also made the TAT system create the workbook with conditional formatting applied to the cells, so that notes less than the passing mark of 4.0 are flagged in red. To handle missing notes (for example, when students were absent), I use a default value of -100 as I did not want to expose non-scientific users to the joys of "Not a Number". This is sufficiently negative to cause obvious problems throughout the calculations.

### 3.3.8 Exam results

In order to shred individual exams that TAT system assumes that the notes are stored in spreadsheets named with the convention EXAMDATE\_EXAMNAME\_Notes with extension either .ods, .xls or .xlsx. The files for a single course should be

---

<sup>6</sup><https://www.tug.org/applications/pdftex/>

grouped together in a subdirectory with the same name as the course name. Each spreadsheet should contain a cell in the first column containing the keyword "Student", followed by a column of student-ids. There should be a cell in the first row containing the keyword "Note", and the notes are in the corresponding intersection of the student-id row and the note column. These are notes for an individual exam, and so should be rounded to the nearest multiple of 0.5.<sup>7</sup>

So an example for an exam filename might 20Apr2020\_ExampleExam\_Notes.ods found in the directory .\1indfb01.

### 3.3.9 Databases

In the Genevan *plan d'études* one third of the second year is devoted to databases (ten out of thirty contact hours). This project would have been excellent opportunity for me to practice planning, creating, updating and querying tables programmatically: the course lists, the comment table and the progress table lend themselves naturally to being stored and treated as tables in a database (indeed, that's why I named them as tables).

Using a database has a number of advantages over a system based on flat files, including:

1. handling scalability and concurrency,
2. data organisation and structure,
3. providing automated backup and recovery programs,
4. offering user authentication and enhanced data security, as well as
5. data consistency based on ACID transactions.

The first four reasons are less important for small, individual projects such as this masters. The last reason, however, would have been useful: ACID stands for Atomicity, Consistency, Isolation, and Durability. Consistency promises that the databases is always in a valid state, even if the system crashes during an update. The TAT system does not make this promise with flat text files: if the system only adds the first few letters of a course name in the raw comments file, for example, the TAT system will not be able to restart without a manual intervention.

---

<sup>7</sup>Individual exam notes are given to the nearest half except in very special cases. One such exception is for controlled exams with an external jury: where the teacher and the jury agree to disagree, the exam note is the average (rounded to the nearest tenth) of the two notes (rounded to the nearest half).

```
comments_file_BROKEN.txt
```

```
d01Apr2023
c1indfb01
+Harry gives conclusion of Pythag
+Harry, Ron correct calc litt for  $$(a+b)^2$$ 

c2ind
```

In spite of these advantages I decided to leave the TAT system based on flat files. First, there is the cost and risk of migration (although the risk should be minin

### 3.3.10 Data privacy

<https://www.ge.ch/organisation/protection-donnees-transparence>

*"Les institutions publiques genevoises sont soumises à la LIPAD, s'agissant du traitement des données personnelles (art. 3 LIPAD). Elles doivent donc respecter les dispositions prévues par cette loi dans tout traitement de données personnelles."*[PPDT18, p. 1]

*"Ainsi, ne sont en principe pas soumises au RGPD les situations suivantes... L'instruction publique accueille des élèves qui résident et/ou ont la nationalité d'un Etat membre de l'UE, sans avoir fait de promotion sur le territoire de l'UE."*[PPDT18, p. 3]

## 4 Choice of programming language

### 4.1 Previous personal experience

I have experience programming in a number of languages : I used **Basic** and a bit of **assembler** as a child, then **Delphi** (Visual Pascal) as a teenager, next **C++** as a grad student, and then I used **Java** and **Matlab** extensively when I worked in industry. I tried **Scala**, **Clojure** and **Julia** during my years transitioning to teaching, and more recently during my GymInf studies I have also had to learn **Python** and **Prolog**.

In general, apart from Prolog, these languages feel quite similar. Some are more weighted towards object-oriented programming (Delphi, Java and Scala) but in general the virtual machine can be modelled as storing the program as a text file and executing the instructions line by line. Variables are declared and assigned values, and expressions are evaluated using iterative rules. Except for Basic and Assembler, which still allow Dijkstra's nemesis the GOTO command, the virtual machine has control of the general flow of the execution using *functions* or *events* (the latter are in some sense just functions owned or controlled by objects).

If we ignore naughty tricks, the programmer can only use switch statements, if-elif-else, switch, or case, to adjust the execution path at a very local level.

The other distinction between the languages is whether or not they are *functional*. Functional languages do not allow the redefinition of a variable, such as Prolog, Clojure and Scala.

TODO discussion of functional languages.

I considered the following factors when choosing the language for my project:

- my current proficiency,
- my interest for improving my proficiency,
- ease for finding solutions to coding problems,
- readability of the code,
- maintainability of the code,
- future applications of improved coding abilities.

There are other factors which did not affect my decision at the start of coding : for example, creating the GUI (graphical user interface), deployability of the code, the speed and reactivity of the program, or whether the code would be scalable. I will comment on these oversights in the conclusion.

## 4.2 New languages

Alongside **Scala** (syntactic sugar over Java with a functional feel) and **Julia** (Matlab maths functionality built directly on C++), the other *new* languages I considered using were **Swift**, **Clojure** and **Rust**.

TODO discussion of these new languages: why they were created, their advantages and disadvantages and possible

## 4.3 Python

At Collège Rousseau, where I teach, we decided to teach the students Python. This was a difficult decision, with each language having advantages and disadvantages. Like the vast majority of Secondary II schools in Switzerland, we decided that the low barrier to entry, the wide use of Python in industry, the teachers' current abilities, the large support community, and the focus of readability narrowly outweighed the use of weak (duck) typing with beginners.

The teacher is expected to have a strong background in the coding language being taught :

- for the pedagogical benefits of "live coding", modelling how to create code for the students to discuss and learn from, TODO citation needed

- creating well written code for students to study,
- for ease of marking and correcting (lots of) student code,
- for spotting mistakes in student code during class time, in order to suggest hints, and
- for recognising bad coding practices and explaining to students the potential pitfalls and how to avoid them,
- for understanding error messages.

## 5 Test driven development

### 5.1 Building IT systems

### 5.2 The waterfall method

### 5.3 Reducing the distance between clients and developers

### 5.4 TDD

### 5.5 And next...?

## 6 The graphical user interface

### 6.1 Choice of language

### 6.2 Implementation style

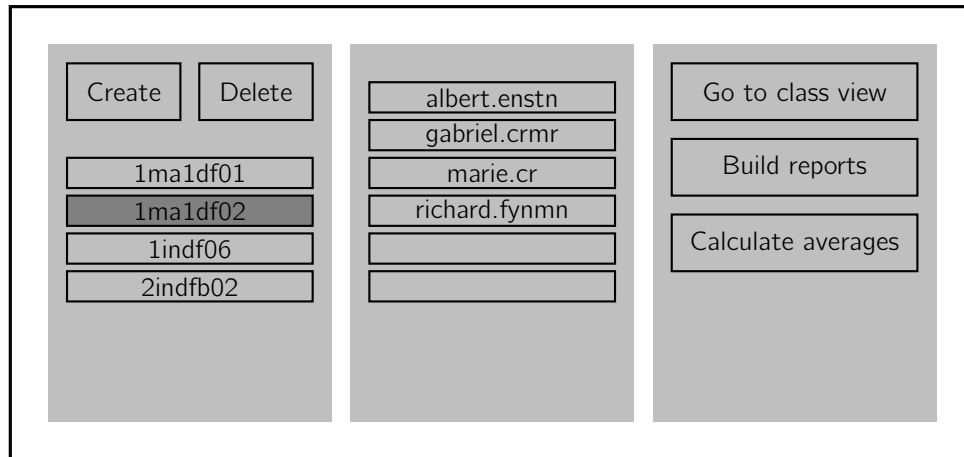
### 6.3 Views

A classic GUI structure is to use *views*, different screens designed for different use cases.

TODO: citation needed.

I based the GUI views on the three phases defined when reviewing the types of actions in section ??: configuration, teaching, and planning. I wanted to be able to use some of the planning actions while teaching, for example suggesting students for interaction, so I moved these to the teaching view. This meant "planning" only left those actions directly related to proposing seating plans, so the "planning" phase is really only "planning the seating plan". Thus I have called the respective views the **control view**, the **class view**, and the **seating plan view**.

### 6.3.1 Control view



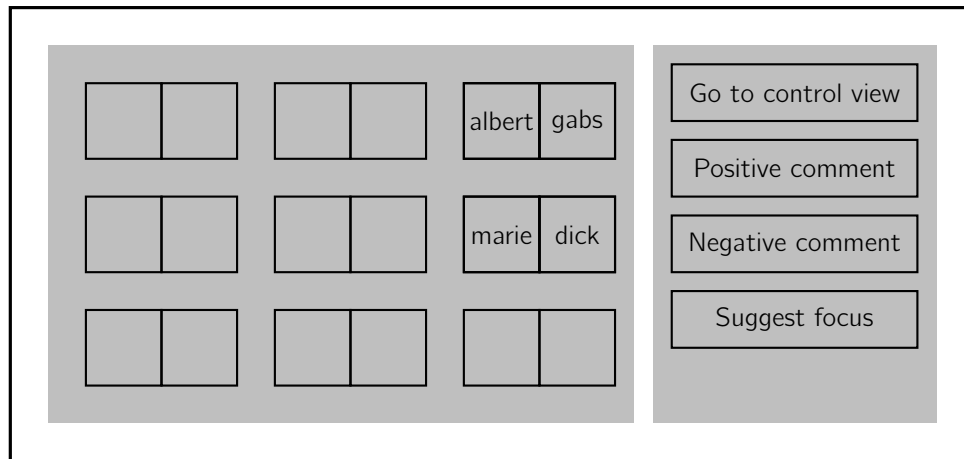
Possible events controlled from this view:

1. Create a new class.
2. Select the active class.
3. Delete the active class
4. Edit the class list.
5. Pass to the class view of the active class.
6. Build student reports.
7. Calculate average notes.

### 6.3.2 Class view

The class view includes the seating plan, both as a reminder for names and as a visual cue for interactions.





Possible events:

1. Pass back to the control view.
2. Change the seating plan.
3. Select/ deselect a student.
4. Record a positive comment for the selected students.
5. Record a negative comment for the selected students.
6. Suggest students for next teacher-student focus.

## 7 A single executable file

Ideally, the TAT system would run from a single executable file. The user would launch the TAT application (for example, by double clicking on the icon) and then all interactions would happen through the GUI.

I tried using two different packages: **PyInstaller**<sup>8</sup> and **cx-Freeze**<sup>9</sup>. Both packages attempt to build standalone executables from Python scripts by bundling the application and dependencies into a single package.

Importantly, both scripts are only able to build packages for the hardware and software used when building the package. This means that **I must build the executable using the school computers**, as the executable I make locally will only run on machines identical to my own. This is a clear disadvantage with the decision to use Python, a scripting language. If I had implemented the TAT system

<sup>8</sup><https://pyinstaller.org/en/stable/>

<sup>9</sup><https://cx-freeze.readthedocs.io/en/latest/index.html>

using Java, for example, then I could have assumed the school computers used a compatible virtual machine.

Unfortunately, at this time I have not managed to automate the process of creating a standalone package. PyInstaller has trouble linking to the pygame module, and cx-Freeze dislikes subfolders. Instead I manually create the executable as follows:

1. Create an empty folder.
2. Copy all the TAT python scripts and the `requirements.txt` into this folder.
3. Create a virtual Python environment for this directory and add the necessary python modules. This can be done at the command line using  

```
pip install -r requirements.txt
```
4. Again at the command line in this working directory, run cx-Freeze  

```
cxfreeze -c gui.py --target-dir dist.
```
5. Copy the `GUI_files` directory into the new `dist` subfolder.
6. From the `dist` subfolder, zip the following files into a single archive ready for distribution:
  - The application file `TAT.exe`.
  - The directory of packaged files `lib`.
  - The directory of TAT options and helper files `GUI_files`.

## 8 Instruction booklet

### 8.1 Page 1: introduction

### 8.2 Page 2: control view

### 8.3 Page 3: class view

## 9 Missing features

Simple executable file

instruction booklet

exam buttons

export seating plan to latex to pdf

handling students who leave

changing given names after comments already made about them

seating plans

using the data to make predictions about students. For example: their behaviour in class has changed dramatically (for the better or for the worse); students who might need extra support in order to pass the year; students who should consider accelerated learning programs. There are lots of resources available to the students and to the teachers, and it is difficult for young teachers to suggest

Automate the process of creating a standalone package, which is currently manual.

## 10 Conclusion

## References

- [Amman16] Ammann, P., & Offutt, J. (2016). Introduction to software testing. Cambridge University Press.
- [Anaya18] Anaya, M. (2018). Clean code in Python. Packt Publishing
- [Beck03] Beck, K. (2003). Test driven development by example. Addison-Wesley.
- [Bloom79] Bloom, B. S. (1976). Human characteristics and school learning. McGraw-Hill.
- [BCSW10] Borba, P., Cavalcanti, A., Sampaio, A., & Woodcook, J. (Eds.). (2010). Testing techniques in software engineering: Second Pernambuco Summer School on Software Engineering, PSSE 2007, Recife, Brazil, December 3-7, 2007, Revised Lectures (Vol. 6153). Springer.
- [Bryant20] Bryant, J., Heitz, C., Sanghvi, S., & Wagle, D. (2020). How artificial intelligence will impact K-12 teachers. Retrieved May, 12, 2020.
- [BS09] Bucheton D. & Soulé Y. (2009). Les gestes professionnels et le jeu des postures de l'enseignant dans la classe : un multi-agenda de préoccupations enchâssées.
- [Coad91] Coad, P., Yourdon, E., & Coad, P. (1991). Object-oriented analysis (Vol. 2). Englewood Cliffs, NJ: Yourdon press.
- [Hat12] Hattie, J. (2012). Visible learning for teachers: Maximizing impact on learning. Routledge.

- [Jorg17] Jorgensen, P. C. (2017). The craft of model-based testing. CRC Press.
- [Mart08] Martin, R. (2008). Clean Code. Pearson.
- [Mart17] Martin, R. (2017). Clean Architecture: A Craftsman's Guide to Software Structure and Design. Pearson
- [MM78] McCorskey, J. C., & McVetta, R. W. (1978). Classroom seating arrangements: Instructional communication theory versus student preferences. *Communication education*, 27(2), 99-111.
- [MSB11] Myers, G. J., Sandler, C., & Badgett, T. (2011). The art of software testing. John Wiley & Sons.
- [Norr98] Norris, J. R. (1998). Markov chains (No. 2). Cambridge university press.
- [PPDT18] Le PPDT - Guide pratique RGPD à l'attention des institutions publiques genevoises (2018).
- [Rob19] Robinson, M. (2019). BOSCARD: A Scoping Tool for Lean Continuous Improvement Projects. In *Global Lean for Higher Education* (pp. 181-195). Productivity Press.
- [Swei15] Sweigart, A. (2015). Automate the Boring Stuff with Python.

## A Original project proposal

Bucheton and Soulé have described the act of teaching as a multi-agenda game of postures requiring good preparation and excellent micro-decisions [BS09]. In their model, they identify the crucial roles of the teacher in controlling the cadence, the atmosphere, the scaffolding and the relationships during the class, and how each supports the learning objective. And yet many of the teacher's most time-consuming tasks do not take place in the classroom [Bryant20]: good preparation and strong follow-ups (for example auto-reflection, marking and parent-teacher interactions) work to support and complement the pillars identified in the student learning process.

Many of these tasks are ripe for automation. As Sweigart writes, “many people spend hours clicking and typing to perform repetitive tasks, unaware that the machine they're using could do their job in seconds if they gave it the right instructions” [Swei15]. For my thesis project of the GymInf formation, I intend to build a suite of tools to support a range of teacher tasks including

- organizing seating plans  
*to aid the atmosphere and relationships*
- suggesting teacher-student interactions for upcoming classes  
*to foster relationships*

- building individual student reports  
*to regulate cadence and scaffolding, as well as for follow-ups*
- creating spreadsheets of marks  
*for follow-ups*

Teachers will interact with the tool using a GUI: it should be easy to use for all teachers, not just those teachers who are computer literate.

I will implement this project using *Test Driven Development*. This is a style of software development that grew out of the 'Extreme Programming' philosophy of the 1990s, encouraging quick development cycles and active feedback from clients. The first book written on the subject, *TTD by Example*, by Kent Beck, is still the main resource [Beck03]. Kent emphasises the need to thoroughly test the requirements in order to have confidence when refactoring or making changes, as well as the need to make incremental changes (both for confidence and for time to market). TTD programming follows the following cycle:

**Red.** Write a failing test for a requirement.

**Green.** As quickly as possible, write code to pass the test.

**Refactor.** Clean the code, removing duplication and renaming functions.

Coding is driven by testing. In the thesis I will explore testing in more detail.

## B Useful quotes

*"The best climate for learning is one in which there is trust. Students often don't like to make mistakes because they fear a negative response from peers. Expert teachers create classrooms in which errors are welcome and learning is cool."* [Hat12]

*"Since it has been reasonably well established that student affect toward a class is related to student learning, student attitudes toward classroom arrangements are a matter of no small concern when determining a choice of classroom arrangement."* [MM78]

<https://github.com/zedr/clean-code-python#table-of-contents>

*"Technology is touted as being able to reduce the time students and teachers spend on menial tasks, time that can be used in other, educationally more meaningful activities. However, there are conflicting views on what is meaningful."* [?, p. 11]

*"Successful education systems typically have absorptive capacity, including strong school leaders and confident teachers willing to innovate. Yet often seemingly trivial issues, such as maintenance and repair, are ignored or underestimated."* [?, p. 17]