Test Plan

Document metadata: LO2 2.1 2.2 2.3 2.4 EVI 1

Contents:

Requirement Priorities and Needs for Adequate Testing

- This section discusses the priority I put on the requirements and what they need in order to be adequately tested.
- The A&T needs result in either tasks [T] which need to be considered in the project schedule, or in tasks which can be directly resolved (e.g. clarifications of requirements; there I directly provide solutions). This is in the spirit of the example test plan we were provided with.

Requirement ID	Priority; Justification	A&T Needs
R1	**high**; the integrity of the database and the data it holds is fundamental to the core functionality of the system.	[Functional Decomposition] We need to know what the UI consists of: - There are two UI components, a user can interact with: - Review platform - Embedded flashcard component - This indicates that we can **divide** analysing and testing this requirement into two tasks, one for

		the review platform UI [T1] and one for the embedded flashcard [T2]. - Both tasks will require similar a methodology which suggests they could be scheduled fairly close to each other, but we have the flexibility to not do so which can prove beneficial since both UI components might be developed independently from each other and at different stages. [Functional Decomposition] We need to know how the UI interactions are propagated to the database: - [Info for T1] The review platform is connected to the database as it can register and login users via the `/api/auth` endpoint of the system, with the user following the respective links. - [Info for T2] The embedded flashcard component can write new review history to the database if an authenticated user clicks on any of the two answer buttons. [Clarification] We need to know what establishes a user of the system to be **authorised**: - A user who is authenticated (i.e. registered and logged into the system) is authorised. [Research] As the test planner (me) is not familiar with End-to-End testing, for testing this requirement we will need to conduct some research on which tools are adequate for this testing approach [T3].
R2	**low**; while the API is an important part of the system, a DoS-attack on it is unlikely to happen in the application domain.	We will need some sort of code or tool which performs mock DoS attacks against the system API [T4]. - We need means to analyse the code/tool's results so that we can interpret them correctly and understand how well the system is protected against this kind of attack [T5]. - [Considering various analysis techniques] Achieving these means could be in the form of research (e.g. having a form of categorisation ready so that if we get result **X** we know that the quality of protection is in category **Y**.) Alternatively, there might be some form of code which automatically performs the analysis. - In conjunction with evaluating the risk of this attack happening, this is a form of risk analysis.
R3	**low**; while it is important	[Measurability] If possible, we need to make this

	to make the system accessible to a wide range of people (especially if the learning experience of the students depends on it), this is not a critical requirement upfront to get the system up and running. It should be realised eventually, however.	requirement **measurable** to increase its **testability**. - We did this for LO1 to make facilitate choosing testing approaches. See 'measurable' attribute for this requirement in the LO1 document were we specify testing approaches. [Reduction] We need to simplify testing this requirement. We see that the document from LO1 proposes two testing approaches, manual human testing [T6] and static code analysis / scanning [T7]. - The principles of Chapter 3 (Y&P) indicate we should consider at least two different T&A approaches if the requirement has high priority. - This requirement has low priority so we might want to consider to only implement one of the testing approaches. Hence, we will only do [T6]. [Research] Ideally, we would use some kind of testing framework to guide us through the manual human testing process to produce useable results which are easy to interpret and analyse. We need to research this [T8]. [Use Case Analysis] Finally, perform the test and analyse its results [T9]. This is a form of use case analysis since we use priorly conducted research [see T8] to identify use cases of how visually impaired people interact with UI.
R4	**moderate**; availability of the system is crucial. However, as discussed for LO1, the ability to compile is not the strongest indicator for system availability, so it is not highly prioritised.	[Measurability] If possible, we need to make this requirement **measurable** to increase its **testability**. - We did this for LO1 to make facilitate choosing testing approaches. See 'measurable' attribute for this requirement in the LO1 document were we specify testing approaches. - We need a way of testing for compilation ability [T10]. - The project is still under development, and even after the release there might be iterations of it we may work on. Constant changes to the software fragments suggest that we need a way of **continuously** testing this, and integrating the compilation ability test [T11].
R5	**high**; as we established with the prioritisation status of R1, the	[Clarification] We need to know what API routes exist and which ones should be auth protected. - There are four endpoints: - `auth` for

	integrity of the database is crucial. A functioning authentication protection for the API protects to database integrity.	registering and logging users into the system - `prompts` for retrieving contents of a single flashcard - `promptcontents` for bulk fetching multiple flashcard contents at once - `userprompts` can retrieve flashcards the user needs to review, and can write new review history to the database. This is the **only route that should be auth protected**. We need to know the possible combinations for testing this: - The API route parameter can be one of the four routes specified above. - The user-status parameter can be either `authenticated` or `not authenticated`. - There are 4^2 possible combinations which means that we can plan to exhaustively test for all cases. [T12]
R6	**low**; while a fast database connection is desirable to satisfy users and therefore improve the learning experience with the system, this is something that could be realised in later iterations of the system.	We need some form of synthetic data to simulate a real-world use of the API [T13]. An important factor constraining testing this requirement is network connection as the system uses a hosted database.
R7	**high**; this is a case where with increasing time which passes from introducing a faulty scheduling unit to detecting and fixing the fault, the cost of the fault rises considerably. Since the scheduling unit relies on past reviewing data, faulty past data will influence future data. Therefore, in the case of a detected fault, the entire reviewing data might need to be reset, something that should	[Boundary Value Analysis] We need to know the range of possible input values for the unit so we can constrain the testing process. We might want to programmatically generate relevant input values [T14].

	happen sooner than later if it's required.	
R8	**low**; consider the case where the scheduling unit is not efficient. Then, it would be unlikely that it would affect availability, reliability or the correctness of the service the system provides. It would only decrease the quality by leading to slower loading times which will eventually propagate to the end-user. Furthermore, the scheduling unit can be anticipated to be fairly simplistic so that a somewhat efficient approach is likely to be chosen to begin with.	We need relevant input data for which we test the performance of the unit [T15] with which we then perform the performance test [T18].

Scaffolding and Instrumentation

This section describes the scaffolding and instrumentation that is necessary in order to analyse and test adequately.

- **Instrumentation:** Instrumentation code refers to code that is added to an existing project to measure its performance or behaviour. This type of code is used to gather data on how the system is functioning, such as measuring response times, tracking user interactions, or logging system errors.
- **Scaffolding:** Additional code needed to access and test certain parts of the system. For example, you might want to unit test that your code deserialises json into objects correctly, but don't want to actually get the json from the server because your unit tests shouldn't rely on a server connection. So some additional setup/mocking is required to essentially simulate the server connection.

- R1

    - We need to write scaffolding to interact with all of the UI components in the E2E test (i.e. to log-in a test-user so that we can access parts of the system UI which are restricted to authorised users) [T16].
- R2
    - The inspection of this requirement does not require any scaffolding (API is directly available and testable) or instrumentation (measuring outcomes does not require internal system monitoring, this will be part of the API mocking tool we will use).
    - Only in the case where the source of poor performance for this test is hardly traceable, we might require some instrumentation code to identify this (e.g. logging activity code). For now, we do not need to consider this in the project schedule as the untrace-ability is only a hypothetical scenario which we assess to be unlikely.
- R3
    - As this is manual UI testing involving human testing, the tester will interact with the system as is. This means that scaffolding code is not required.
    - We might require some instrumentation to validate that UI interactions are correctly interpreted by the system (e.g. when a user selects a button via the keyboard and the UI shows so, we might want some form of internal system confirmation, that this button is registered to be selected) [T17].
- R4
    - No scaffolding needed.
    - In the case that the compilation test fails (i.e. system does not compile), we would like a report on why the compilation failed. This is already provided by the compiler, hence no instrumentation code needed to measure this.
- R5
    - We need scaffolding to enable the test to test for registered users. We can recycle scaffolding from task [T16] for this, so this does not need to be considered in the project schedule.
    - We do not need additional instrumentation as the API endpoint responses are already implemented in a way that they can serve as instrumentation.
- R6

    - Writing to the database is constrained to authorised users. We can recycle the scaffolding from task [T16] to enable any performance tests to simulate the writing process.
    - Additional scaffolding required includes a synthetic data generation tool (see task [T13]).
    - Instrumentation is likely to not be needed depending on the tool we are going to use for performance testing this (e.g. JMH, Apache JMeter, Gatling).
- R7
    - Scaffolding: we might want a method which generates relevant inputs instead of manually typing them, especially if the space of relevant inputs is big. This is part of [T14].
    - Scaffolding: the unit works with `QueryDocumentSnapshot<DocumentData>` objects returned from the Firebase database response which have a build-in `toDate()` function that we utilise in the unit. It would be difficult to create such objects in our test suite, hence some scaffolding is needed for the unit to also accept other representations of dates [T19].
    - Instrumentation: not needed.
- R8
    - It is likely that this will be a somewhat minimalist test setup. However, if needed we can recycle scaffolding from task [T14] to generate relevant inputs for this performance test.

Task and Test Specification

This section provides a specification for tasks [T] identified above, including tests.

To purposefully assign the tasks identified above to the appropriate lifecycle phase in the next section, we estimate how long the task will take, and what is necessary for the task to be possible: resources, code, data, results from earlier tasks. A&T needs are already specified above, here I only specify additional needs regarding results from other tasks.

Test tasks are colorised.

Task ID	Description	Needs	Estimated Completion	Pass Condition

			Time (ECT)	
T1	perform an E2E-test on the review platform UI to test for R1	prior completion of T3, T16	2h	no authorised access is gained
T2	perform an E2E-test on the flashcard UI to test for R1	prior completion of T3, T16	2h	no authorised access is gained
T3	research basic concepts of E2E-testing, and tools to realise them		0.5h	
T4	find suitable tool which performs mock DoS attacks to verify R2		0.25h	
T5	prepare means to analyse results from simulated DoS attacks		0.5h	
T6	perform manual human UI accessibility testing	prior completion of T8	0.5h	the user can navigate all elements via keyboard
T7	We removed this task from the test plan as it was found to be non-critical for now.			
T8	research testing framework for human UI accessibility testing		0.5h	
T9	analysis of T6	prior completion of T6	0.25h	
T10	test for		10min	system compiles

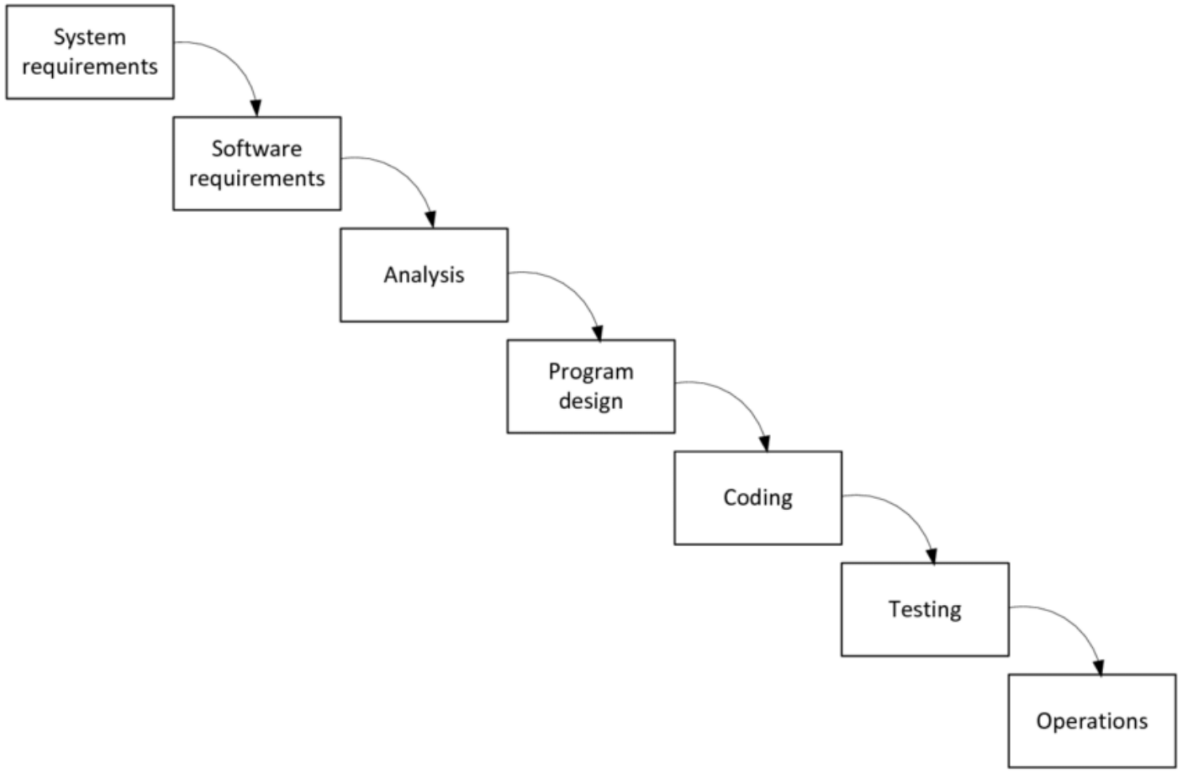
	compilation ability to verify R4			successfully
T11	integrate the compilation ability test into a continuous testing environment	prior completion of T10	1h	
T12	exhaustively test for authenticated access of API routes	prior completion of T16	2h	can only get authenticated API access if user is authorised, otherwise not
T13	synthetic data creation tool		0.25h	
T14	generation of relevant input values		20min	
T15	creating relevant input data for the unit, **unit testing**	prior completion of T14	10min	unit returns expected result for relevant input
T16	scaffolding to programmatically simulate registering and logging in a user, so that a driver or other part of a test program achieves the needed authentication status		0.5h	
T17	instrumentation code to log how UI interactions are interpreted by the system		0.5h	
T18	performance test		10min	performs

	of scheduling unit to verify R8			<100ms
T19	scaffolding for the scheduling unit to accept different date formats for testing purposes		10min	

Process and Risk (Lifecycle Assignments)

This section demonstrates my ability to work out where to put a testing activity into a lifecycle and therefore my capacity to develop a comprehensive test plan.

We choose the Waterfall cycle. This choice is obviously somewhat unrealistic but sufficient for demonstrating the required skills for LO2.



Waterfall cycle schema. Note that this diagram should be cyclic, i.e. there could be an arrow from Operations to System requirements. Source

Considering the waterfall lifecycle, we follow the “V model” (Figure 2.1 from Y&P, shown below) in a sequential manner, beginning unit testing only as implementation

commenced following completion of the detailed design phase, and finishing unit testing before integration testing commences.

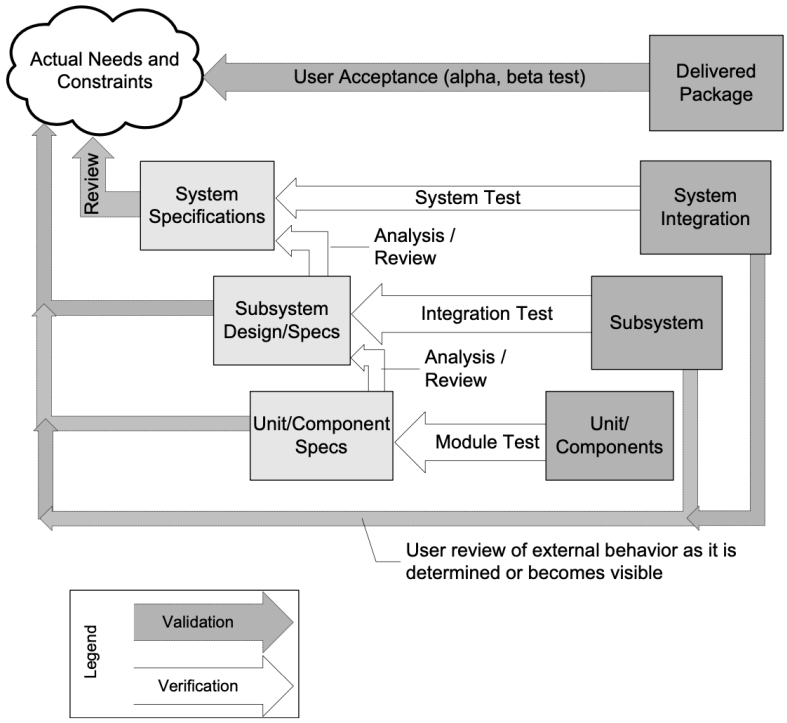


Figure 2.1: Validation activities check work products against actual user requirements, while verification activities check consistency of work products.

All of above yields these assignments:

Analysis

Tasks assigned to this phase usually build on testing results from previous lifecycle iterations. This means that including them early on in the lifecycle will allow to adjust activities in the following phases accordingly.

Assigned tasks: T6, T9

Design

For the design phase the focus is typically on creating a detailed design of the system, including specifications for the architecture, components, interfaces, and data structures. However, it is important to consider testing needs during the design phase, as it can be more difficult and costly to add testability features or address testing issues after the design is complete.

This justifies putting research tasks here as their results might produce additional design constraints we need to include early on during this lifecycle.

Assigned tasks: T3, T4, T5, T8

Coding

I put tasks involving instrumentation and scaffolding here because I anticipate that implementing the during the coding phase of the project would be most time efficient as the knowledge of how the (just completed) relevant software fragment would still be fresh and facilitate the completion of these tasks.

Assigned tasks: T13, T14, T16, T17, T19

Testing

Unit before Integration before System, as justified above.

- Unit: T15, T18
- Integration: T12
- System: T1, T2, T6, T10, T11

Risk Assessment / Evaluation of the Test Plan

We consider the risks mentioned in chapter 20 (Y&P):

- Personnel risk: any contingency that may make a qualified staff member unavailable when needed.
- Technology risk: risks introduced by certain technology used by the project.
- Schedule risk: schedule risk arises primarily from optimistic assumptions in the quality plan.

Personnel risk

- **Risk:** there is are two specific instances of personnel risk:
    1. Regarding the implementation and execution of all tasks: with me being the only developer for this project, it would be fatal if I became unavailable.
    2. Regarding tasks involving human participants (i.e. T6), they might become unavailable.
    - **Mitigation approach:**
        1. If I became unavailable due to unforeseen reasons (e.g. illness), this would affect the project schedule. As a mitigation strategy, I would apply for Special Circumstances to extend the time available for this project.

2. Have back-up participants (myself included) ready.

Technology risk

- **Risk:** the authentication COTS components do not meet quality expectations.
    - **Mitigation approach:** Include COTS component qualification testing early in project plan; introduce alternative COTS components prototyping exercises.
- **Risk:** the testing tools used for DoS attacks, API mocking and E2E testing do not meet expectations.
    - **Mitigation approach:** Anticipate and schedule time for training with new tools; have alternative tools ready.
- **Risk:** the quality of data produced via our synthetic data tool does not meet the required standards. Unrepresentative synthetic data could cause potentially serious issues for project design.
    - **Mitigation approach:** Include a task in the schedule which is solely dedicated to ensuring high quality data output which meets our requirements for using this data for testing.

Scheduling risk

- There is a high risk that the time cost for most of the tasks is underestimated, especially regarding tasks involving the construction of scaffolding and instrumentation code. This will largely depend on the quality of the system's software and design, and therefore how easy these tasks will be. This may be amplified by the chosen lifecycle model as scheduling delays can easily accumulate due to the rigid and somewhat unrealistic nature of the model. Therefore, we can identify three risks:
    - **Risk:** Inadequate unit testing leads to unanticipated expense and delays in integration testing.
        - **Mitigation approach:** Track and reward quality unit testing as evidenced by low-fault densities in integration.
    - **Risk:** Inadequate integration testing leads to unanticipated expense and delays in system testing.
        - **Mitigation approach:** Track and reward quality integration testing as evidenced by low-fault densities in system.

- **Risk:** Inadequate system testing leads to unanticipated expense and delays for moving onto the Operations phase in specific cycles.
    - **Mitigation approach:** Track and reward quality system testing during an adequate validation activity.

Conclusion

The test plan identifies relevant tasks and offers an adequate lifecycle approach for them. However, we evaluate that the use of a waterfall lifecycle bears high scheduling risks which the project leader should be aware of if they decided to follow along with this plan.

Implementation of Instrumentation of Scaffolding

Relevant tasks: T13, T14, T15, T16, T19

- T13
    - First, we need to define the structure of the data we want to create. In this case, the data will be used for mocking the system API.
        - For the `GET` endpoints of all API routes, we only require a parameter consisting of one or multiple flashcard `ID`s.
            - Their structure is a randomly-generated alphanumeric string of length.
        - There is only one exposed `POST` endpoint for writing to the database, one for the review history.
            - This requires an additional `remembered` field with a boolean value.
    - The following data structure will suffice:

```
synthetic_data = [
  {
    ID: alphanumeric string of length 20,
    remembered: True | False,
  },
]
```

    - We want to be able to specify the amount of data, so every ID-remembered data instance we call `instance`. The `instance` parameter will be an input for how many instances we generate with the scaffolding script.

    - The code and samples of synthetic data are in the `LO2/T13` folder.
- T14
    - We want to generate relevant input values for the flashcard scheduling unit. First, we need to identify:
        - What kind of input should the unit accept?
            - (The unit can be viewed at `website/utils/scheduler.js`)
            - Looking at the code, we deduct that the unit works with a `results` data structure (an array) of which each element has a `calculatedNextDue` field with the due date. This is the structure of the input data we want to generate.
        - What makes an input value **relevant**?
            - We can observe that the unit has three main logical control flows, with an additional one on the third path.
            - We will want to generate input which goes through all possible paths (exhaustive **path testing**).
            - We can see that the unit does not have any kind of error handling for invalid inputs, so this does not seem to be required to test for.
            - We deduct that the following inputs are relevant:
                - `result` of length 0 (to cover the first path)
                - `result` of length 1
                - `result` of an arbitrary length, ordered.
                    - Here we observe that only ever the two first elements matter for the unit, so we can stick to the length of 2.
        - The range of relevant input values is constrained and we can generate them by hand. The results are documented in the `LO2/T14` folder.
- T15
    - We can build on the results from T14.
    - Since this is a performance test, we add more relevant inputs such as a very large `results` object.
    - The results are documented in the `LO2/T15` folder.

- T16
  - The easiest way to achieve this is to create a user account which is solely for testing purposes. Its credentials can then be safely read from a secret environment file. This means we need to:
    1. Create the test user.
    2. Add to secret environment file.
    3. Provide a scaffolding method to login and establish an authenticated session. This involves research as to how the COTS authentication components expose a login API.
  - The results are documented in the `LO2/T16` folder. For security purposes I am not releasing the test user credentials in public.
- T17
  - Logging in JavaScript can be done via the `console.log()` method which is available for all browsers.
  - The entire UI only consists of a few buttons for which we can simply change the `onClick` attribute.
  - It would be somewhat redundant and unnecessary to copy and paste all the changed code into a dedicated `LO2/T17` folder, but an example of how this instrumentation code shall be implemented is provided.
- T19
  - Extended the scheduler to accept and use other date formats.

Evaluation of the Instrumentation and Scaffolding

After having implemented it, we provide an assessment of the adequacy of the instrumentation and scaffolding at this point.

- T13
  - **Adequacy:** The synthetic data differs from the real data since it only specifies two data fields. The real data has additional fields. However, this difference is neglectable as the unit we're using this data for testing purposes for does not take any additional data fields into account. Therefore, the synthetic data is adequate.

The data generation tool is adequate as well as it can be customised how much data is generated, a desirable attribute we specified earlier.

  - **Potential improvement:** We could extend the generation tool to include the other additional fields in the data generation. This would make this scaffolding data more generalised which might prove useful for other testing purposes later on.

- T14
  - **Adequacy:** The generate input values are adequate for testing this specific unit. However, one should note that dates represented by a `yyyy-mm-dd` formatted string. The database uses datetime objects so this is a discrepancy between the synthetic scaffolding inputs and the real inputs. However, the provided scaffolding is still adequate for testing purposes as it is easily possible to programmatically match the different date representations during testing.

    This scaffolding is essential as we do not want the testing of this unit be dependent on actual data retrieved from the database. This would introduce unnecessary compound factors such as database integrity.

  - **Potential improvement:** One could standardise how dates are represented across the entire system so that discrepancy matching will not be required anymore.

- T15
  - **Adequacy:** Adding the larger `results` object seems trivial now but is actually non-trivial as there could always be changes applied to the scheduling unit which would then not perform well. A larger input is indeed what is to be expected for the system, so this is an important and useful addition of scaffolding.

  - **Potential improvement:** none found, this is a fairly simplistic case of scaffolding.

- T16
  - **Adequacy:** The provided scaffolding is adequate for testing purposes as it allows to programmatically create an authenticated user session.

  - **Potential improvement:** It would be nice if any created test data could be cleaned up after the testing. While the data does not interfere with the running system, the data would eventually accumulate with every test run.

- T17
    - **Adequacy:** Logging to the web console is ok and adequate for what we are going to use this instrumentation code for (manual human UI accessibility testing).
    - **Potential improvement:** An alternative approach (which involves more time, which I do not have for this task at this stage), ideally I would create a log well-structured file which could be used for automated analysis purposes later on.