Reflection and Traceability Report on SubLiMat

Uriel Garcilazo Cruz

1 Changes in Response to Feedback

1.1 SRS and Hazard Analysis

The SRS documentation was the first document of its type I ever made for any of my programs. The interpretation of the requirements was a bit difficult at first, especially for the lack of specific details in the problem statement, attempting to maintain a balance between being too specific and too vague. The theoretical models that served as foundation to later documents were also one of the most difficult parts to write. The feedback received from my peers and instructor pointed at clear semantic issues between functional vs. non-functional requirements. To this end, the detailing of specific condition boundaries for my theoretical model (i.e. edge cases) and its refactoring into the return of a function with mathematical notation was feedback that arose later in class.

Table 1: Feedback Response Tracking - SRS

Label	**Changes/Issue**	**Commit**
SRS and PoC	Initial submission for SRS and PoC shared with peer reviewers and instructor	aecf366
SRS - Instructor feedback	Changes in the definition of theoretical models, and non-functional requirements	ab39de
SRS - Issue: Please declare edge cases	Added handling and documentation for edge cases as per peer domain expert's suggestion	ab39de
SRS - Unclear IM1 Definition	Instance model updated, adding clarity and refactoring th. model as function	ab39de

1.2 Design and Design Documentation

The module interface specification and the module guide for SubLiMat started with a complex design with 9 modules, from which only 5 were implemented at the end. My understanding of such documentation was limited (still is due to the extensive nature of the topic). Especially confusing at the beginning were the multiple definitions of modules, whose categorizations seemed at times arbitrary or justifiable only from a functional perspective. The slides from the course helped, as I was able to focus this categorization in libraries, control, abstract, abstract data types and record. From an operational point of view, and as a coder, having a better understanding on how abstraction in modular design differs from the definitions of modules given by the programming languages themselves gave me a missing piece in the capacity to scale and expand my projects. I began the files with a clear notion of functionality, services and secrets. The effects in the quality of my code became obvious during testing, where this modularity allowed me to evlauate the code efficiently. My understanding in modularity was further reinforced by the instructor's feedback, whose comments were primarily oriented towards the type of information that should be included given a specific module type.

Table 2: Feedback Response Tracking

Label	**Changes/Issue**	**Commit**
MIS/MG Submission	Initial submission for MIS/MG	a2789ce
MG Correction	Initial submission for MIS/MG	fd2bd6f
MG New version	Update version with newer version in modular design	3ada846
MG new version annotated	Minor update reflecting the new version of documentation for April 1st.	fb7bb98
MG New version	Update version with clarification on data types and 5 modules	2cf0fbd

1.3 VnV Plan and Report

The validation and verification plan was one of the most technical documents in the project. I found it easy to write in principle, but difficult given how easy it was to overshoot the goals to achieve, based on the limited amount of time, and available human resources to help in the development of the software. The difference between validation and verification, which addresses very different perpectives in the development of the software (Are we building the right product? vs Are we building the product right?), was one of my biggest take-aways from building this project. As it turns out, my project cannot compete with the heuristic approach that most molecular biologists use nowadays to analize genome-size data. In this sense, the VnV plan could only be justified as an exercise. However! Although the assumption for the course was that validation tests passed (even if we didn't implement them), I do believe this project sets the right track for building the foundation for future releases, which could greatly benefit from the structure written for SubLiMat. In addition, the problem it's intended to solve — What is the best substitution matrix to use for two sequences of DNA — relies on the assumption that a global alignment is achievable, something that most heuristic algorithms cannot deliver. Therefore, I find the services delivered by my software to have validity in the field of bioinformatics, albeit limited to a specific set of sequences.

Table 3: Feedback Response Tracking

Label	**Changes/Issue**	**Commit**
VnV Submission	Initial submission for VnV	b57dbb0
VnV Correction	Minor update reflecting the new version of documentation for April 1st.	98663bc
VnV Updated version	Main update reflecting the new version of documentation for April 4th after comments.	9dbf902

2 Challenge Level and Extras

2.1 Challenge Level

The challenge level for my project was **basic**.

2.2 Extras

Due to limitations in time and resources (which was pointed at me during the VnV plan), the main extra delivered by my program is a **user guide** and a **live demo**.

3 Design Iteration (LO11 (PrototypeIterate))

The development in the design and implementation of SubLiMat was iterative and sequential. The general progression of the project began with the SRS documentation, which enabled a high level of abstraction in the program elements. How to handle the specifications in a timely manner was the focus of the VnV. It was a this stage where I needed to go back to the SRS after addressing the comments of my domain experts. An updated version of the SRS that declared clear edge cases for the theoretical model, and a more detailed description of the non-functional requirements allowed me to refactor the modularity of the program. The MG and MIS documentation reflected those changes in their initial version — with 9 modules — vs its final version — with 5 modules. The necessity for such changes was highlighted by the lack of secrets and/or services in some of the modules, which were necessary for the program to run, but didn't encapsulate enough complexity, any state variables nor services to justify their existence as modules. In addition, some of the most important changes became clear only after diving into the Proof of Concept, where I was able to receive feedback from potential users. After their considerations, it became clear the end user was feeling restricted by the accessibility of substitution matrices in the program, with the obvious solution of adding a file manager module. That's why the final version of the program inserted a control module, and a file manager module.

4 Design Decisions (LO12)

The design decisions made during the development of SubLiMat were based, at least initially, on maximizing the readability of the code, and in such a way the architecture of the program could reflect the MG and MIS documentation. However, some of the most important decisions in the design of the software relied on the responsibility to encapsulate the services and secrets reflected in design documentations. In particular, the file manager module could have been dissolved, with its processes assigned to each of the abstract data types that

needed to communicate with the operative system, by reading the files or saving the data. However, the decision to keep them all encapsulated in the same specialized module made the code more readable and easier to maintain or change. From this experience I learned that building programs for scalability begins with separation of concerns. If my program had had the need to communicate with more modules, changing the behavior of each module would have resulted in a cumbersome task.

In addition, I was particularly interested in designing the control module because it detailed the flow of information in the system. Although I wasn't able to develop the program further in the time available, I would like to explore a future release of the software where the control module is changed into an interface or abstract data type, enabling its integration into larger systems, and the file manager module would be raised at a higher hierarchical level, whereas the alignment module would become a Decision Hiding module, allowing the user to explore other pairwise alignment algorithms.

5 Economic Considerations (LO23)

There are no economic considerations for the implementation or use of SubLiMat. Its use is strictly intended for research purposes, and the code is open source. This is why SubLiMat includes an MIT license with a Commons clause condition that restricts the software from commercial use.

6 Reflection on Project Management (LO24)

I was lucky to have chosen a project that wasn't too complex, as compared with some outstanding projects shown during presentations. This gave me the opportunity to handle changes in modules and specifications in such a way my project didn't cascade into complex rearrangements in the software design.

6.1 How Does Your Project Management Compare to Your Development Plan

The following table, 4, has been modified from [**?**], and summarizes the progress of the project, and how it compares to the development plan.

6.2 What Went Well?

Because there were no heavy dependencies used in my program, the installation process was straightforward, and SubLiMat was able to be installed in most live or demo versions of online python interpreters. The use of GitHub was also a great help in the development of the project, as it allowed me to keep track of the changes made in the code, and to share it with my peers.

Task	Status	Followed status	Assigned to	Start Week
Planning		**Low**		**Week 1**
Conduct research on topic	Completed		Uriel Garcilazo	
Draft project problem statement	Completed		Uriel Garcilazo	
Gather stakeholders	Non-Completed		Uriel Garcilazo	
Plan tentative goals	Completed		Uriel Garcilazo	
SRS Analysis		**High**		**Week 3**
Define business objectives and goals	Non-Completed		Uriel Garcilazo	
Formalize project requirements	Completed		Uriel Garcilazo	
Define priorities	Completed		Uriel Garcilazo	
Verify Document by Primary reviewer	Completed		Junwei Lin	
Verify Document by Professor	Completed		Smith Spencer	
VnV Plan		**Medium**		**Week 6**
Define Unit tests	Completed		Uriel Garcilazo	
Verify Document by Primary reviewer	Completed		Junwei Lin	
Verify Document by Secondary reviewer	Completed		Junwei Lin	
Verify Document by Professor	Completed		Smith Spencer	
Design Document		**High**		**Week 9**
Proof of concept	Completed		Uriel Garcilazo	
Define Modules	Completed		Uriel Garcilazo	
Identified Syntax and Semantics	Completed		Uriel Garcilazo	
Verify Document by Primary reviewer	Completed		Junwei Lin	
Verify Document by Professor	Completed		Smith Spencer	
Development		**High**		**Week 10**
Development	Completed		Uriel Garcilazo	
Execution	Completed		Uriel Garcilazo	
Test cases	Completed		Uriel Garcilazo	
VnV Report		**High**		**Week 12**
Report test cases	Completed		Uriel Garcilazo	

Table 4: Project Progress Tracking

6.3 What Went Wrong?

Due to my inexperience handling latex files, filling the initial documentation for SRS was tremendously challenging. I found caveats and compiling issues every 5 minutes. Finding the references and getting used to the .bib format was equaly frustrating. After VnV, however, I was much more comfortable with the syntax.

A more theoretical issue related to my documentation was a clear documentation of the input system. Because of the difficulty handling latex previously stated, I was reluctant to major changes in my SRS documentation, although it was becoming clear that giving the end user the opportunity to handle the substitution matrices on their own, rather than being hardcoded in the program, was an important feature. By the time I reached the proof of concept, the change was unavoidable, because delaying the change would have had serious drawbacks down the MG and MIS documentation.

6.4 What Would you Do Differently Next Time?

I would pick a project capable of receiving more input from the user. Rather than building the architecture in Python and terminal, I'd like to try building an API that users and classroom peers could use to give me feedback. This would make the gathering of data from a unique set of highly specialized users much more feasible, as they wouldn't need to install anything.

7 Reflection on Capstone

I learned:

- There are multiple tests, and some rely on principles of natural selection and biology
- Experienced programmers have a hard time developing testing
- Specifications and requirements are the heart of a good project (i.e. meant to be subject to expansion and change)
- Continuous integration allows for the automation of testing and code quality
- There are utilities like pylint, black and isort that allow for standardization of the code

7.1 Which Courses Were Relevant

UBC's CPSC110: Introduction to Systematic Program Design

7.2 Knowledge/Skills Outside of Courses

I had to learn about packaging my modules, setting up standardization for PEP8, and most importantly,learning latex and mathematical symbols.