# NCLVerse

A digital universe for student learningPrathyush Pramod NStudent Number110505992SupervisorDr. Phillip Lord

## Abstract

A university student has various digital tools at their disposal to stay on top of their academics. Unfortunately, these tools suffer from a lack of vision when it comes to their architecture. Their lack of conceptual integrity in metaphors, knowledge representation and user interface behaviours results in an incoherent whole that fails in realizing its full potential.

This dissertation is a proposal for creating a modular ecosystem of apps that forms a unified platform sharing a common visual language with consistent knowledge representation. I propose that creating such a system results in increased ease of use from the user perspective, better awareness of student performance for the teachers and saner software development at the developer's end owing to the modular, centralized and coherent nature of the system.

## Aims

Architect and engineer a modular ecosystem of applications that form a unified platform to aid student learning.

## Objectives

My objectives are four-fold in this project:

- To research and summarize the current systems in use and analyze their advantages and disadvantages and retain all attributes that are deemed to be desirable.
- To investigate about befitting architectures for digital ecosystems and on fostering communities around them.
- Iteratively develop the applications and devise an appropriate architecture for interoperability between them.
- To evaluate my design decisions based on feedback from colleagues and observation conducted on prototypes through focus groups.

# Table of Contents

# Acknowledgements

The ideas presented in this dissertation are synthesized from various approaches that have been proposed over the years in software industry and other active fields of research.

I drew my inspirations from the teachings of great many people which include my tutors, colleagues and friends. Many specific ideas have emerged from many informal conversations I have had with friends, blog posts I read and conference talks I watched most of whose names have escaped me. But attribution has been duly noted wherever I could identify them. I express my gratitude towards everyone who have helped shape my thinking and approach towards software development.

My views about software development has been greatly transformed by works of Alan Kay, Alan Perils, J.C.R Licklider, Seymour Papert, Joe Armstrong and Rich Hickey and their principles on software development. Their ideas have helped me form the bedrock for shaping my views toward creating software. I am deeply indebted to the Clojure community for producing quality software and instilling in me some of the best philosophies produced in the software community. A more comprehensive list of all the technologies used can be found in the appendix.

My ideas about metaphors and conceptual integrity come from George Lakoff, Mark Johnson, Fred Brookes and Donald Norman.

I am grateful for the work of Edward Tufte's work on data visualization which served as a source of inspiration for the work in unifying visual interfaces.

I am deeply indebted to my tutor, Phillip Lord for allowing me to embark on his journey. His advice on software development and time management turned out to be of great value for the project. The code authoring tools he has developed for my code editor of choice Emacs made code annotation a far smoother experience. His continuous support has impacted both the tangible and intangible aspects of the project in more ways than one.

It was a great thing having my friends and colleagues to discuss the dissertation with at different stages of the project. I would like to thank Benjamin Brown, Moad Abouzamazem, Alexander Filipov and Achint Soni for their support throughout the project.

All code developed towards creating this ecosystem has been open sourced and I look forward to see what comes of it.

## Overview

The dissertation is divided into 6 parts inclusive of the introduction and the code. In the upcoming sections, I detail about various facets of the project. Theory section describes meta-level ideas that formed conceptual grounding for the dissertation. Readers would be able to see how these abstract ideas were made concrete in method section. The discussion section details how individual apps were put under scrutiny of the conceptual framework derived and their internal as well as overall consistency within these framework is analysed. Towards the end, I evaluate my dissertation based on the success criteria stipulated, describe my take aways from the project. I leave with pointers on what I think future directions of this project might be if further work is to be carried out.

# 1 Introduction

The aim of this dissertation is to construct an ecosystem for learning that is architected from the ground up to form a cogent platform that students can leverage. This would be accomplished with the help of a suite of apps that together aid them in their learning process by employing a coherent visual language and behaviour throughout the apps in the system. The central problem addressed is to minimise the endemic problems that accompany systems integrated in an a la carte fashion. I propose much better can be done by providing a conceptual framework and constructing apps out of this minimal set of principles put forth.

## 1.1 Background

The preliminary conception of this idea came into shape when I started using digital systems enlisted in my university. I started using these at a time when I was being exposed to the core principles of Computing Science and I felt a major disconnect between the architectural principles of these apps with principles that I was learning in Computing Science. For example, when the deadline of a course work or the timetable of a lecture is changed, it gets updated on a system known as NESS and an email is sent out but Blackboard another system used for tracking student progress shows the unmodified deadline. This results in generating confusion and discomfort among the students. The students who were oblivious to the email sent out miss this fact entirely and end up getting to know about this change only on submission time.

During my time at university I started building a few apps in my extra time to help my colleagues on different problem domains. But overtime I found out that my own apps suffered from the same kind of systemic errors found in the academic environment. But this lead to the realisation that this is a recurrent problem that can be solved by imposing an overall architecture. Hence this thesis is an attempt at unifying all those applications I developed over time under a single architecture and growing it into a single unified platform with consistent knowledge representation. Software systems found in a modern academic environment lack a coherent architecture. Status quo is to pick modules à la carte that works well in a stand-alone fashion but not so well with other applications in the environment. This creates problems that are insidious such as duplication and erroneous data, incoherent user interfaces and so on. To make this concrete, Newcastle University currently employs a set of digital tools for catering to the digital side of academics: Blackboard, NESS, Outlook, Panorama, Timetables, Crypt and S3P. No two of these share the same visual language or architecture. This results in the students end up having to learn a new interaction every time they use a new product. This learning remains non-transferable. This often works to the detriment of the user as it results in confusion, reduced productivity, erroneous and redundant data which all leads to faulty decisions when trying to interface with these tools.

## 1.2 The Solution

As a student who spent the majority of his life dealing with digital systems and have seen others go through the same difficulties, it felt worthwhile to choose my last year Bachelors degree dissertation to come up with a solution to this problem. I propose a solution based on my conviction that designing an environment with a unified user interface and consistent knowledge representation throughout, would enable the student to stay on top of their academics, stay connected with their peers and teachers, and to a certain extend, help inform where to spend their limited time and focus budget on.

There are a lot of other additional benefits that results from adhering to this methodology. For an academic practitioner as the end user, this platform can be tailored to benefit them with better data collection about the student performance and increased interaction with the students with the help of software created for the ecosystem. By following to time proven software development techniques like modular architecture and single responsibility principle, I argue that it also simplifies managing complexity that arises from software development.

Now that I have outlined what this project is about, let me describe what the focal points of this project was not. I have increasingly noticed that bespoke constructs and context specific performance optimizations most of the time compromises flexibility. This means that it was in a sense worthless to go for specific performance tuning when the final architecture hadn't been finalized. This is not to be taken to mean that they are de-emphasized because they were irrelevant but it is only because of the nature of the project where the end requirements were not available prior to development, it was most reasonable to keep them flexible and build composable abstractions that are as general as possible which helps to solve a wide range of problems. Focusing or improving upon a certain facet of a software means that if that component gets unused in the end, all the effort goes waste hence a this project adopted a lean methodology of production popularised by Toyota [TPS 1988] having in mind the core idea that effort expended towards this project were least wasted.

Having described the background that lead up to the creation of this project let us see the metaphors and and principles that underlie the creation of this system in the next section.

## 2.1 Metaphors of the Ecosystem

This chapter details the research made into metaphors that were used to model the system in abstract. Metaphors employed here are primarily devised so as to imbue a coherent meaning for the whole project. As such, it represents general background knowledge that informed decisions made throughout the project. They formed the grounding for the architecture derived which in turn influenced and justified the design decisions made. Throughout the rest of this thesis, I'll be invoking these metaphors to explain why certain trade-offs were made. Here's how they formed the hierarchy that influenced the decisions made throughout the project:

[Metaphor informs architecture informs design principles inform implementation.]

### 2.1.1 Metaphors We Live By

A metaphor is typically defined as a literary device used for suggesting resemblance of one thing to another. But George Lakoff and Mark Johnson in their seminal work 'Metaphors We Live By' [MWLB 1980] advance the idea that our metaphors are pervasive not just in language but in our thoughts and actions. They argue:

**"Our ordinary conceptual system, in terms of which we both think and act, is fundamentally metaphorical in nature. The concepts that govern our thought are not just matters of the intellect. They also govern our everyday functioning, down to the most mundane details. Our concepts structure what we perceive, how we get around in the world, and how we relate to other people. Our conceptual system thus plays a central role in defining our everyday realities. If we are right in suggesting that our conceptual system is largely metaphorical, then the way we think, what we experience, and what we do every day is very much a matter of metaphor."**

Thus a metaphor has the power to drive the conceptual understanding of a user. Computers and the processes inside them have been compared to a lot of metaphors in the past, one that I think that gets closest to the real nature of the device is one by Alan Kay [CS 1984]. He compares it to the ultimate protean system, one that can take any shape or form. He writes:

**"The protean nature of the computer is such that it can act like a machine or like a language to be shaped and exploited. It is a medium that can dynamically simulate the details of any other medium, including media that cannot exist physically. [sic] It is the first meta-medium, and as such it has degrees of freedom for representation and expression never before encountered and as yet barely investigated. "**

This remains true even today since the fuller capabilities of the computer hasn't been fully understood or explored yet. It follows that a computer can be programmed to fit virtually any context or use case. The one that I think most befitted this dissertation is to visualise the world as two. I view user facing end of the software which includes the user interface, information architecture and everything that the user interacts with as a universe. This is where the name NCLVerse is derived from. The other world which realizes this universe is that of computations. This is modelled using a centralized client server architecture.  These metaphors were responsible for informing the why behind certain decisions.

### 2.1.2 Building an experiential gestalt

Lakoff and Johnson also introduces an idea known as experiential gestalt in their book. They contend "that a cluster of components are experienced as a whole by human beings which they find more basic than the parts."[Lakoff Johnson 1980] This directly references the work of a prototype done by Rosch [Rosch 1978] in her work on categorizations done by humans. A recurrent complex of properties, our concept of causation is at once holistic, analysable into those properties and capable of a wide range of variation. My aim in structuring this system has to enable a coherent whole whose sum is different when put together than the individual parts. This has been realized with the help of recurrent patterns carefully positioned throughout the ecosystem.

Though it might look like these models were figured out foremost and the truth is quite the contrary. The apps were retrofitted in the end when the final architectural fit was realized. The architecture had to be grown along with the work and it was only with frequent stepping back and revaluation that I was able to figure it out as it stands now. This emerged only after a lot of experimentation and constantly evolving the architecture over time to accommodate the discrepancies. Even then the architecture as it currently stands is not to be held as absolute and constant pruning and controlled evolution is required to adapt it to newer apps and accommodate evolving user needs. This means that the design guidelines and models outlined for software development in the next sections are to be evolved along as new apps are designed and never take them to be steadfast rules.

### 2.1.3 The Two worlds

The products delivered for this project encompass two different worlds: The user facing interface of the system and the underlying world of computations that enabled this. The nature of these two systems, I have come to realize over the course of the dissertation are very different.

[Insert picture here]

This is a visualisation of how I model the two systems: The upper layer being the front end user facing side of the underlying structures. The dissertation will talk about how the ideas and concepts of the first world are more cogent than the logic layer that lays hidden beneath. It can also be reflective of where my strengths and weaknesses lay.

### The Visual World

The first world is that of the user facing end of software. This comprises of everything visual as well as the not so visual (UX) part of the system. While visual side of software allows for infinite imagination it is constrained by the underlying world of processes constraints it because of the limits of the computation.

-
- Orbits as Courses
-
- Modules as planets
-
- Solar system as stages
-
- Departments as galaxies
-
- Galaxies form Universe
-

The view adopted here is that of a universe and the student an astronaut. I visualise each orbit as a stage of a course that student progresses through. A set of these courses can be taken as a solar system and a composition of these solar systems form a galaxy which has been mapped on to a department. All the academic faculty of the university can thus be thought of as a composition of a large number of galaxies. Inorder to make this concrete, Computer Science can be thought of as the galaxy in which G400 is a solar system in which modules are planetary objects that rotate in their course. This is where the name of the project is derived from.

## The Computation World

While the user interface of the systems takes the shape of a universe, these are built on a substrate of complex underlying systems where things are very fragile if care is not supplied. This is because of the nature of software development which is ever changing and could potentially lead to breakdown of whole software ecosystem if they are ever allowed to unmonitored changes. Though the software development methodologies presents many ways to organize this complexity, the basic nature of software breaking on interaction with it is still incumbent. This is partly because Computer Science is an emergent field which is still not fully understood by humans, while various methodologies have been proposed to solve the software crisis (Refer Out of Tarpit and Formal Verification Here) no single solution has been proved to be superior.

Given the case it was of much importance to hedge against the brittle nature of software and the environment of constant flux software work against. After evaluating these criteria the model that seemed most fit was to use the model of a centralized client server model.

Rosen and Shklar in their book Web Applications Architecture [Shklar Rosen 2009] describes the server model as:

**[In a client-server model], servers ... execute by waiting for requests from client programs to arrive and processing those requests. Client programs can be applications used by human beings, or they could be servers that need to make their own requests that can only be fulfilled by other servers. More often than not,**

**the client and the server run on separate machines, and communicate via a connection across a network.**

This model albeit in a centralized fashion with a single server at the heart of the services that acts as a data store with all the other applications acting as clients to this service is the model this thesis propose.

The central data store for communication is named Ivory Tower (another name for Tower of Babel from the Bible). This abstraction is the core data store from which the programs acquire their data over wire and execute respective computations. It is to be noted that even though all the current applications in this ecosystem are written in a single language, this architecture affords polyglot development and diverse styles of programming. This was one of the primary decision drivers behind building the central abstraction known as the Ivory Tower which can be enabled to transfer data in any message exchange format.

Thus these two worlds, universe on the user facing side and a centralized server and clients model in the developer facing end of this ecosystem are the two metaphorical lenses through which the design decisions framework adopted for this thesis make sense.

### 2.1.4 On the dangers of metaphors

Like any metaphor, the ones adopted here conceal some facets of the idea while revealing others. Leslie Lamport has an incisive essay on why the metaphor of biology is limiting in the sense that it gives an amorphous definition to programs that is inherently complex and not understandable, in this sense, metaphors we use to understand the systems can be insidious. As mentioned in the essay, the metaphors we adopt to describe the systems might take us to a cul-de-sac.

Full evaluation of the drawbacks of these metaphors are has not been fully determined due to the limited time scope of the dissertation. Healthy skepticism is to be maintained whenever an idea or a new concept is introduced into the system. As mentioned earlier the models are not to be held absolute and the need for excessively shoehorning a new application into the system is quite possibly revelatory of the inadequacy of the model adopted, a better approach would be to evolve the current conceptual model so that it can accommodate them. This is described in the next section which details about the architecture.

## 2.2 Growing the Ecosystem

People are central to any software architecture. The ecosystem would not be complete without a supporting documentation of all the functionalities. The ecosystem as it stands now can be recognized as a set of core apps and supporting structures around it. These supporting structures are vital for fostering communities around the ecosystem. This prose elucidates the kind of supporting structures that have been built and analyses rationale behind the creation of these structures from the viewpoints of the end user.

The methodology chosen towards this thesis is to tackle it from the viewpoint of a human centered process to foster a community around these systems. My views on building software was greatly transformed by a short treatise written by Richard Cook called How Systems Fail [Cook 1998] The author suggests that software doesn't exist in a vacuum, it is made alive with the people who use the software to meet their ends. He describes that they adapt the system to maximize production and minimize accidents.

In this sense, it was of foremost importance to value the expectations of the end user and the kind of actions enabled for them by constructing the systems in certain ways. In this process I have identified personas of 3 users among many.

First one: The student Second one: The tutor Third one: The developer

The core focus of this project has been the student and is tackled in detail in the upcoming sections. Hence I will focus on the other two groups and detail in brief how the design decisions made from an architectural perspective adds value to the lives of a faculty member and a developer.

For a faculty member, this ecosystem if extended would enable them to collect student progression datasets in a much smoother way. If appropriate application programming interfaces are put in place, there is a huge potential to structure the ecosystem in a way that it optimizes to collect data on various aspects of student learning. A cursory implementation that can be done with the help of the current set of apps proposed is to learn about interaction of the student with the course materials using the Curriculum app proposed in the discussion section.

From the perspective of a developer, the major advantage over current systems is that all the apps are developed in the open in a modular fashion. This means that reuse of the components developed for these apps can promote code reuse. Moreover, the ecosystem also enables creation of bespoke APIs that takes care of a single facet of student learning. This means that even developers other than the core team under control of these systems (students, faculty members and the like) can build their own applications using the interfaces provided.

### 2.2.1 Support structures

I have identified that without supporting structures most of the apps however well designed will fail to meet the user expectations. Whereas, when supplied with enough documentation, it helps the user leverage these apps to their full potential.

There are four documentation websites created in order to provide support the NCLVerse. Two sites are dedicated for the developers and designers of these systems, there is a comprehensive documentation of all the core applications in

NCLVerse for the potential user of the applications and finally a website that hosts this dissertation itself. They are as follows:

### 2.2.2 Documentation Website

The documentation website details about all features of all the apps that have been built for NCLVerse. A feature that has not yet been implemented but not too hard to achieve is to have a contextual help menu. What this means is that, whenever a user wants help about a certain interface element, she can click on the help button and then choose the part that she is confused about and it would redirect her to that part of the documentation website where the interface element would be mentioned in detail. This can be implemented with the help of hyperlinks that map the interface element on to fragment identifiers on the documentation website.

### 2.2.3 Code Website

This is done in Literate programming style in websites generated with Clojure. Literate programming style is an approach to programming introduced by Donald Knuth [Knuth 1992] in which a program is given as an explanation of the program logic in for the human reader as opposed to the machine. This can be thought of as a narrative account of the developer's understanding of how the code works.

The ecosystem would not be complete without a supporting documentation of all the functionalities. This is implemented using a Clojure app.

The app is a blogging platform where developers can log in and authenticate with the systems, obtain their license keys etc.

### 2.2.4 On juggling the two worlds

The real implications of creating the two world is felt when the users are brought into the system. As I demonstrated and observed the usage patterns, it disabused me of many notions I had held over the software and it informed on where to drive the software.

As software construction goes it requires the frameworks to be internally consistent and follow some contracts. Much like how a function deals with an input and chains it on to another. But when a developer starts working with this, he is faced with the challenge of keeping everything intact.

Hell are other people's abstractions. Picking libraries became an exercise in building taste on what is palatable and if you identify with the author’s ideologies.

### 2.2.5 Supporting Websites

I have devoted a about 1/5th of my time developing this dissertation towards developing these support structures since they were a vital aspect for the students to learn about the new technologies in the ecosystem. One of the drawbacks of spreading things around is that there a lot of points that are not in the picture. It can be argued that a monolithic app would have one single point and all the elements can be found out if you explore the same app. But when spreading the apps around. This is done in the current ecosystem by consistently cross-linking the picture with the. The dashboard mirrors the concept of a central platform and then reaching out from it.

## Notes to Self

Software as catalyst for improving time user spends online.

## 3.1 Analysis of Current Systems

## Competitive and Similar Products

## An observation

Some people go on to refresh the browser which can be simply eliminate with an email notification which user can enable when they want to be updated or this data can be shown on the Dashboard using websockets if they prefer. This is detrimental for the server which takes redundant hits and for the user since he is pointlessly clicking on the reload button when there's no actual update. This induces a lot of time consuming behaviour such as constantly refreshing the NESS (current portal for getting marks at Newcastle.) This in my opinion is serious time that can be spent elsewhere in a more productive manner.

## 3.2 Implementation

Premature optimization is the root of all evil.

Late binded partly because that fits with the style of my working and party because this enables more things to be grasped.

## Introduction

Looking at the definition, we find A web application is a client-server application that uses a web browser as its client program. It delivers interactive services through web servers distributed over the Internet (or an intranet). A web site simply delivers content from static files. A web application can present dynamically tailored content based on request parameters, tracked user behaviors, and security considerations.

The main focus of the dissertation is not to innovate or break new grounds in algorithmic analysis or one up performance benchmarks. It is simply to find out if an ecosystem of apps works better for the student experience over a unified system plethora of services that are put together without any kind of conceptual integrity.

## On work

This approach is popular and is known as EDSL.

As with most new ideas, it originally happened in isolated fits and starts.

We would only know where we are going after we reach there.

Looking back with a rear view mirror.

Optimizing after the fact.

Premature optimization is the root of all evil.

How things can be made more abstract when boredom strikes.

## Candidates

Low fidelity prototypes

High fidelity prototypes

I have sticked with a methodology of following very minimal low fidelity prototyping. One reason is that I have a good grasp since I worked in the user interface industry for around 7 years now and the other reason being, while the low fidelity prototypes do provide the affordance of being able to iterate through a large mass of designs it ends up being very far from the final product.

## Improving interactivity

The main impact of AJAX is that the web experience is brought closer to that of client-server applications, allowing for a more immediate dynamic interaction between the user and the web application.  AJAX and Websockets

Things are in place to poll the differences. If the system is put to practice enabling websocket polling is not far away. Clojure libraries such as Sente [link] enables this in a very effortless manner.

## 3.3 Designing the Visual Language

One of the main challenges in creating software products is developing the visual language. User interface design is a cross functional discipline that encompasses areas such as interaction design, visual design and information architecture. When developing a design language, it is of utmost importance that harmony is maintained across the whole spectrum of interface design. Harmony, both internal as well as across, is to be conserved at a macro level with a coherent visual language and obvious information architecture, down to the micro level where consistency among knowledge representation artifacts such as icon labels, module codes and colour indicators come into play. Discrepancy among any of the variables spread across this spectrum would result in reduced usability of the end product. A successful act of balancing in these results in a user interface which is easy to access, understand, and facilitate user activity. These principles are encapsulated by creating a design philosophy which employs a common lexicon of terms to describe the various concepts that occur throughout the system.

### 3.3.1 The visual language

The school of thought in user interface design recently bifurcated with the advent of so called flat design. [Allan 2012] [Schiff 2015] Both comes with their own precepts on how a design aesthetic is to be envisaged. Combining my previous experience in the industry as a user interface designer and in the light of detailed analysis of the two schools, the approach that I adopted here is a synthesis of ideas from both worlds. It is an optimum which can be found between the two styles by appropriating the styles to the right context. The approach adopted in this project is as follows: The apps use shaded look that is characteristic of the old design aesthetic for all interactive elements. This is done with the idea in mind that all elements that are pseudo-3D in their appearance would attract user interaction since they indicate a sense of depth which is characteristic of real world objects. [Apple 2002] This is juxtaposed with a flat aesthetic for elements that are purely graphic visualization of data but doesn't afford any interaction. The logic behind this is that the user would be able to quickly predict which elements are interactive and thus adheres to the principle of least surprise. [Raskin 2002]

A user interface is much more than meets the eye. As a matter of fact, Donald Norman, a pioneer in user interface research outlines a set of principal heuristics on his website that are widely used in the industry to improve the usability of an application [Nielsen 1995]

## Notes to self

Need more support for the usability area.

### 3.3.2 Usability first

ISO9241 [ISO9421] define usability as:

**"The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use."**

Seen in this light, the design language's main goals were in maximizing usability. A more detail analysis is available in the methods section but to give an brief illustration of how design tradeoffs were made, it can be seen in this image from one of the apps designed for this ecosystem called ProgressMate that the irrelevant details were relegated to the back and only the details that mattered were brought to the forefront.

Provide detailed discussion of the example.

One other thing that I have followed throughout is to minimize the requirement of user interaction with the elements on screen and making it intentionally harder when the action is a rare one. Even thought one additional click might only amount to half a seconds delay compared to one, this when conducted by 4000 users doing it 10 times a day results in 4000 * 2 * 0.5 = 5 human hours on a large scale. This amount of time lost only due to an overlooked aspect from the designer's side in alarming. Hence, considerable attention was supplied in putting the relevant data first and hiding away the irrelevant. A detailed analysis would reveal that this principle has not yet been fully achieved yet, but I have strove for such accordances wherever possible.

A corollary that falls out from this decision is that actions that which are not used frequently are hidden behind two step actions. Even though the experience derived by the user is largely visual there are a lot of critical invisible parameters such as responsiveness of a user interface plays a considerable role in determining how the user experience is impacted. When it comes to the visual user interface, the possibilities of representations are almost only limited by the imagination of the designer. Since this project attempts to create a suite of apps directed towards student learning, it was of high priority that two factors were given main focus: Primarily, it had to be ensured that no time of the user was wasted and secondly that there was a coherent narrative spread through the app and how it is all connected together.

## 3.3.3 Coherence over Consistency

The design methodology adopted for this project outlines a unification of the design using a consistent visual design language as proposed by the style guides like that of Google and Heroku [link] [link]. I have found from my experience over the such guidelines while hinders the creativity of a designer. Since what is being created here is a platform I had to ensure that the future is kept in mind and all that this platform sets out is to have overall coherence. That is coherence outweighs consistency.

Where possible consistency has been brought in, but it felt like a better choice to make room for new design elements since a fuller understanding of the ecosystem gained by observing usage patterns after this ecosystem goes into production can only yield understanding at this understanding. Only a preliminary understanding of this has been achieved so far and it was most sensible to leave it open. This allows new design elements to be incorporated into the system at a later stage.

A system was to use a cogent visual language that covered the whole spectrum of the apps. A lot of principles in constructing the systems were borrowed from Design of Everyday Things and the Tufte tetrology. I greatly believe in the concept of personal computing as an amplifier for human reach. And most if not all of the concepts governing the UI has starts flowing having this as the central precept. Inorder to maintain this, the things that had to be done were tight feedback loop. Conceptual Integrity was maintained.  Designing with the user in mind.

Each app has a primary colour and a secondary colour for highlighting. The behaviour is made consistent with the help of user interface patterns like the navigation bar on the top.

[Give a sample image here]

### 3.3.4 Beyond the visuals

There are parameters of the system that something that honours the consistency within the system can be made since there's more than just visual consistency that meets the eye that keeps the system coherent. This is discussed in the following section. Provide detailed analysis of how information architecture has helped define the internal consistency. The discussion section gives a thorough account of the decisions that have been made across the applications in this ecosystem to make the user experience as smooth as possible.

## Extra Reading

http://nxhx.org/RedefiningSoftware/

The case against user interface consistency
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.90.6480&rep=rep1&type=pdf

## Knowledge Representation

http://www.jfsowa.com/ontology/index.htm

## 2.4 Nature of Software Development

Building software is a continuous decision making process. Software is repeatedly hit by wavefronts of changes initiated by multiple agents. Hence, it was of utmost importance that the philosophy and the principles laid out here would not stand in the way of adaptability of the system. This was held as the central principle behind structuring the system or in other words ability to adapt change is the major parameter worth benchmarking the efficacy of this architecture against.

### 2.4.1 Developing the Architecture

Architecture is defined as "the process and product of planning, designing and constructing buildings." Buildings here can be substituted for software without losing much in intend.

**There are only two hard things in Computer Science: Cache Invalidation and Naming Things. - Phil Karlton**

Software systems are complex because of the fact that there are a lot of entities and myriads of relationships among them. Inorder to have a control in creating systems as they grow complex, it was essential that recurrent patterns were spotted and given a name to, so that it helped to manipulate and reason about these concepts. As the size of the systems in the current project started growing one of the biggest challenges came in the form of having to name things. It was one of the increasingly difficult concepts to minimize the name clashes as and keep them separated apart so that there was no confusion.

Fred Brooke's in his book argues that coherent systems are the one's where the whole system can be kept in ones' mind as noted by Fred Brooks. As he notes, it became increasingly difficult when it came to the naming aspect to keep all of this in my mind.

Inorder to construct an architecture, a language haS to be developed to talk about it. This made the architecture tangible and provides an unambiguous description of architectural building blocks as well as concrete system while abstracted away from implementation details.

The architecture required definition of the building blocks (types of things) from which the actual system will be built. It can be thought of as a bespoke language constructed in order to talk about the system. Though I couldn't take it to a fuller refinement, I have provided systems diagrams [provide crosslink] to explain the construction of the software created for the ecosystem.

### 2.4.2 Choosing a programming language (Read)

**Controlling complexity is the essence of computer programming. — Brian Kernighan**

Software development is an enterprise that is deeply enmeshed with controlling complexity has been one of the pivotal concerns in this project. As Djikstra points out "computing scientist’s main challenge is not to get confused by the complexities of his own making", this rang quite true throughout this project.

Almost all of this project was completed in a programming language known as Clojure. It's a Lisp that runs on the JVM. As of the writing it has emerged as the most popular lisp ever written. But popularity can't be held as an indicator of quality. What I feel is a good indicator is the ease with which a tool enables creation and in this regard Clojure proved to be a solid tool to build software systems. I am convinced that the nature of minimal syntax of Lisp and composability of mathematical functions are great concepts to manage the emergent complexity when developing software. The great tooling around the core language and an active and smart community around the language have been pivotal towards this achievement. The lanugage being functional in nature also helped me learn a lot about programming in general since my education at the university was heavily focused towards object oriented and imperative languages. Simply put, it has been transformative in more ways than one. Simple core abstractions, immutability, referential transparency and a whole lot of other great concepts that Clojure entails have greatly shaped the way I program and I hope will continue to influence my future. Implications of developing in a functional paradigm is mentioned in the methods section.

A note that that I have to add about Clojure is that it simply doesn't make the software development process easy for you. It is simply not so for an incoming newbie. You have to be well versed on a large pool of concepts to help it to see the programming domain in the way it sees but once you are beyond that plateau, leveraging Clojure leads to a simplified view of creating processes and procedures.

### 2.4.3 Shape of the systems

Describe the steps taken in order to making the systems legible.

### 2.4.4 Dogfooding (Rewrite)

There is a concept known as dogfooding which means to test the software by yourself. Since I am one among the target audience for which I'm constructing this system for, it was a natural fit. But instead of focusing on how these helped inform my academic decisions which I'm relegating to the evaluation section I will briefly mention how I got a good taste of this when I used my docs-gen [NVDocsGen 2015] [link] to generate the documentation for describing the apps in the ecosystem. It was a good experience to create software that describe the docs generator. This was a good benchmark to test the soundness of the concept as you continuously used the software yourself while also presenting an alternative view, I made sure that I shared my creations with my colleagues to see the sticking points or potentially confusing interactions that might arise.

### 2.4.5 Componentized apps (Reread)

Systems had to be constructed with the idea that specifications would always change. And as the saying goes change being the only thing that is constant, it was high priority to build this ecosystem in a manner that it accommodated change. This has been achieved by keeping the systems modular.

The nature of lisp lends itself to create components which can be weaved together to form a whole app. I could only fare as far as creating structures of web components as outlined in the development section, but some more work is to be done in order to make it truly componentized. This means that apps can be composed as a series of higher order functions.

One of the central precepts of this dissertation is that of modularization and all the apps can be considered as an module that consists of further more modules inside them. Due to my limited time and experience I had with lisp this has not been realized to a fuller extend in this project but to give a brief account of how this can be enabled, here's some very minimal Clojure code that uses the compose function in Clojure.

; Comp(ose)ing the components together (def app ((comp) header main-area footer)) ; Passing in the data (app data)
This would help enable the componentization and hence reuse of different modules developed for each app among the different apps

## 2.4.6 Software eats itself. (Reread and Illustrate)

Better abstractions enable structuring your programs better. Throughout the process I was able to see how a thorough abstraction of a certain model of process enabled me to be more flexible at designing my functions. In order to give a simple illustration:

The code

(def )

whereas if I employ the following code:

(def )

would result in much more flexibility in terms of the actual product. This was proven time and again throughout the time I did my dissertation.

One way is to start from scratch and supply as much care and control over everything by building them from the ground up but the scope of this dissertation as it had to be produced in a limited time didn't allow for such levels of explorations. I have to mention that the language adopted for building this ecosystem played a key part in keeping the complexity to a manageable level. Also, this nature of software being always in flux is what lead me to create the current architecture of centralizing it with the pervasive model of Internet technology - the client server model.

## 2.4.7 Advantages

It was essential to follow certain models and probably I have tried to explore different sets of architectural styles for each apps. For example the ExamVault website follows a typical client server model with most of the visual rendering predetermined on the server side whereas in NCLProgress the majority is done on the client side, similarly for

IvoryTower which forms the backbone of apps by providing them data uses a RESTful architecture in delivering the information. I will be detailing about these styles later in fuller detail later in the thesis but it is to be noted that modularizing the apps and creating a domain for a coherent set of functionality enables the exploration of creating bespoke architectures that are well fitted to each domain. This could be a major win from a polyglot perspective as I have mentioned this enables for integrating newer systems into the ecosystem with ease.

### 2.4.8 Tradeoffs made

### 2.4.9 Towards better architectures

Lack of my knowledge with respect to software architectures in general and inability to visualize the future of these apps have rendered to this to be sub-optimal but the good thing about software is that everything can be rebuilt from scratch once you have explored the domain space thoroughly.

A few structures that I have in mind to architecture the ecosystem was Supervisor-Workers model from Erlang OTP documentation. Attention had to be paid to the behaviour of this software. I have decided that if further work is to be carried out if I wants to create a completely resilient system.  The implications of adopting this model is further explained in the [future] section.

## Notes to self

Fred Brookes' tar pit.  Silos and Snowclones.

## 3.5 On Research Methodology

The methodology followed here is to collect information from various sources. Alan Kay speaks about thinking the present reality as a construct which is a result of amalgamation of different theories that people have forged up and to reconstruct it, look beyond what is and cherry pick the set of consistent ideas that will help you form a harmonious approach.

Following this approach, I have tried to pick only the elements that matters constantly pruning these resources as needed and re-adjusting my hypothesis in the light of new data.

## Caveats

Confirmation Bias. You look where you steer. This has been something that I am not totally sure that I have gotten rid of. If I am to stand back and evaluate what I have done so far in the way I have done it. 80% is because I have chosen the routes and the routes then determined where I will reach. Choosing the path is the hardest and the funny thing I have learnt is that most of the times you will be wrong. But the beauty then again is that life even though feels short gives you ample time to backtrack on your ideas. Especially because this is a domain where in my opinion, humans have the least clue on how to create process that work and to write ones that work, let alone how a coordination of them.

## Research

Was made into user interface designing. The modules that I took in the university guided my decision.

## I do I understand

Experential knowledge as the guidelight. http://blog.ncase.me/i-do-and-i-understand/

## Structure of Clojure Ecosystem

Clojars, Leiningen for dependency management and automating workflows.

## Database Solution

As I started looking for a database solution, one of the foremost and most widely used entrantants in the field was something known as SQLKorma which was written by Chris Granger but was waning under development cycles, this lead me to look for options and a twitter conversation led.

I eventually had to settle with Postgres.

## 4.1 Current fragmented ecosystem

A typical student in Newcastle University is introduced to a nebulous and fragmented ecosystem. It lacks a unified architecture and this results in a lot of time being wasted trying to learn, unlearn and re-learn interactions. At the moment, Newcastle University employs a set of apps listed as follows:

Blackboard: Keeps track of student materials. NESS: University's own interface to keep track of student data and coursework deadlines and submission. Timetables: A web interface to timetables. Library: A dedicated website for library to search for books at various libraries of university, keeps track of loaned books. Outlook: Web based email client developed by Microsoft Crypt: Archive of exam papers

NUVision: Archive of lecture videos.

These are some of the web applications that a student typically comes in contact on a day to day basis. A preliminary analysis of the user interface of these apps reveals that almost none of them share consistent user interaction patterns that help ease the user experience. The only superficial one that can be pointed out is the Shibboleth Gateway which enables access to all the apps from a single point entry.

## Observations

Some people go on to refresh the browser which can be simply eliminate with an email notification which user can enable when they want to be updated or this data can be shown on the Dashboard using websockets if they prefer. This is detrimental for the server which takes redundant hits and for the user since he is pointlessly clicking on the reload button when there's no actual update. This induces a lot of time consuming behaviour such as constantly refreshing the NESS (current portal for getting marks at Newcastle.) This in my opinion is serious time that can be spent elsewhere in a more productive manner.

One another problem that results from the apps not sharing a common database is that there is no apparent semantic separation of concerns between the apps. Blackboard application displays the coursework deadlines on a notification area which is immune to changes done in NESS. This discrepancy results in confusion among the students since there are two incongruous representation of the same data. Each app deployed in the university at the moment takes care of more than one thing with apparently distinct databases.

This dissertation proposes that much better can be done by imposing an architecture that ties together all the different apps with the same visual language. As outlined in the design section, all the apps shown here are designed with core precept of having conceptual integrity internally as well as amongst them. An overall architecture and visual language has been developed to describe all the applications from the ground up.

This problem can be eradicated by having a consistent knowledge repository. My research pointed at two ways to achieve this. One is to have a centralized database that takes care of all the information single handedly. Other approach, which is much more resilient in nature, is to have a distributed database that keeps the information in synchronization across the different applications. This dissertation adopts the former approach of having a centralized architecture since it was of importance that the various requirements and contracts between the data had to be figured out before a distributed database was deployed.

But the major advantage of having separated these apps at their concern boundaries is that it allows for much more focused development. By condensing the amount of information each app holds it drastically reduce the amount of complexity budget the student has to store in her head while navigating a user interface. Unlike Blackboard, which is a melting pot of everything university related, each application in this ecosystem does one thing and one thing only. This endows the user to determine the most salient information faster since the number of choices and user interface acrobatics she has to perform is reduced.

## 4.2 Ivory Tower

**There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies and the other way is to make it so complicated that there are no obvious deficiencies.**

**— C.A.R. Hoare, The 1980 ACM Turing Award Lecture**

Hoare's insightful remark points at two ways to build software systems. I feel applications developed for this thesis exists somewhere in a transient stage between these two extremes. Currently they have a lot of identified and as of yet unidentified deficiencies but in some sense they always will be so at one point or another due to the turbulent nature of specifications that tries to account for a constantly evolving academic context. Here I present all the constructs that I have identified towards addressing some of the concerns that current applications address. By virtue of the single responsibility principle, I have tried to extend functionalities of the current apps wherever it seemed to add value. An example would be the modality presented in one of the applications to follow where the sliders enables the student to explore and evaluate their future coursework marks and its impact on the stage and overall averages.

### 4.2.1 Ivory Tower

As outlined in the theory section, the architecture's central point of contact is the consolidated datastore that I have named as the Ivory Tower. This is an abstraction that sits on top of a database and accumulates data from various dynamic endpoints. It is responsible for serving data to the various components of the universe and also for maintaining a consistent interface for all applications. It does this by providing an API that every app can interface with.

This approach provides some advantages over letting all the applications directly connecting to the database. First of all it decouples the implementation from the interface. This essentially means that any database can be swapped in place of the current concrete representation. This approach has led in my experience to keep the architecture be more flexible and secure.

A student's only point of contact with this system is through the gateway login. This is similar to the Shibboleth interface Newcastle university currently has.

[Image here]

It invokes the metaphor of space mentioned in the theory section with an astronaut helmet in place to give the feel of an exploration. Once the user logs in, they are redirected back to the respective application where the request originated. All the rest of the communication with the application is conducted over the wire with AJAX requests. A popular approach is to use JSON as the data exchange format but development in Clojure enabled me to the use EDN in place of JSON and it made the implementation a bit smoother. One other alternative worth exploring here is the newly introduced data format Transit [Transit 2015]

## 4.5 Exam Sieve

Exam sievew sa conceived in mind to analyze question papers. Now it's mostly voluntary but if studens so choose this effort can be changed to the faculties end so a student get a fuller appreciation of the topic distribution. Sequence of yeras in the UI.

### What?

A store of exam papers analyzed with data visualization tools.

### Why?

Inform students about topic distribution and act as a consolidated archive for exam papers. Provide additional meta data over the existing Crypt exam vaults.

### How?

Enables labelling of question papers.

### When?

Mostly required during exam times.

### For Who?

For students who are facing their exam. Acts as a prioritizer for their exams.

### Design

Why was a sidebar preferred?

Moving sequence of three years to show the revelant years since curriculums get outdated.

Each exam paper is evaluated.

## 4.6 Progress Mate

Progress mate is one of the apps that I consider to help drive the sure decision and it think is the single best illustration of how this platform would hepl provide the user with more leverage than any toll currently rpovides.

Only downward tree accumulation at the moment but in the future, this would be expandend to have predictive powers such as slidering forwardig ving an estmate of how much to achieve.

### Implementation Details

This was implemented with the help of React. It is a new framework that is created by Facebook to create The V of . Clojurescript has Om. A model in mind that helpde me to come to a fuller appreciation of thi was the idea of a tree. One way data flow. Illustrate with a diagram.

### What?

A store of exam papers analyzed with data visualization tools.

### Why?

Inform students about topic distribution and act as a consolidated archive for exam papers. Provide additional meta data over the existing Crypt exam vaults.

### How?

Enables labelling of question papers.

### When?

Mostly required during exam times.

### For Who?

For students who are facing their exam. Acts as a prioritizer for their exams.

## Design

## Warm Colors vs Cool Colors

Warm colors to indicate problem. Cool colors to indicate good marks.

At a view the students can have an idea of which module they are doing a good job at and what needs improvement. Current systems lack any kind of such indication to drive the efforts of the students. Little hints like his can go a long way towards driving the efforts of the student.

## Development

React has things split into different places and  one way data flow.

One image that helped me grasp how React works is to view it as a tree model from development and then coordinate the changes.

[Probably insert an image here describing my understanding here.]

Om manages all the state in one place. It’s an outgrowth of Clojure philosophy and seeing React from a lenticular prism of Clojure philosophy.

## Downward accumulation

Marks are accumulated only downwards.

## 4.3 Curriculum

Curriculm is the app that provides consolidated storage for all the course related materials. All the materials are kept and if possible, implement a search capability using JSON that filters the search result. All the videos which are now available through panorama is now indexed and kept alongside. Use of module icons is employed. The timeframe on tis dissertation only allowed for cerating a limited set of icons but if future work iss to be conducted a more comprehensive iconset can be developed and deployed.

What advantages does it provide over current systems?

A fuller appreciaton of this website would be possible when the accompanying dashboard is brought into context. The current practice uses an indication in the form of a [provide screenshot] clock notification to show that one among many activity has been conducted. Following the DRY and SR principle ensures that much more focus and clarity can be brought about when representing notification for the specific module.

## On implementation

Through the wire EDN from ivory tower. This data is then marshaled. Evaluate why EDN is better than JSON for this purpose.

## What?

One stop place for all degree related course materials. Includes videos, courseworks, course materials.

## Why?

## How?

## When?

## For Who?

## Design

## 4.7 Student Pad

If Dashboard is the nexus of all applications, student pad is the community nexus. There is also plans to transform it as a cross departmental channel communication in the future, but with the limited scope of this dissertation, this has been shortened down to a student only community site that lets the student. At it's current form it meerly acts as a regulatory service which helps students to opine on a module and share their views.

### What?

A store of exam papers analyzed with data visualization tools.

### Why?

Inform students about topic distribution and act as a consolidated archive for exam papers. Provide additional meta data over the existing Crypt exam vaults.

### How?

Enables labelling of question papers.

### When?

Mostly required during exam times.

### For Who?

For students who are facing their exam. Acts as a prioritizer for their exams.

### Design

Why was a sidebar preferred?

Moving sequence of three years to show the revelant years since curriculums get outdated.

## 4.5 Dashboard

Dashboard is the nexus. It is the central hub for tracking down everything that is happening with the projet. THe dashboard yields itself to an interaction :

[Icon Fan Image]

This interaction as outlined in the overview part of this section allows for a consistent navigation pattern to explore all of NCLVerse

Notifications

Will be displayed on the icons this mwould be enabled by using websockets since websockets would allow for doing realtime updates and is much better than AJAX for polling. From a user's perspective this means that they can at a glance get to know where the activity is happening and hence make an informed decision. The ordering of the apps are to be determined in the order of most recent notification. One of the short comings of this view is that only a limited amount of apps can be displayed using this style, even though this can be exteneded with the helpo fo arrows a better extensible pattern is to be formed when the number of applications increase beyond.

Write about mobile considerations.

Another approach was but this was escheewed in favour of the currevt interaction as it seemed like a good metaphorical fit for the student as an astronaut in space metaphor that was invoked earlier.

## What?

The frontend for all activity happening inside the NCLVerse.

## Why?

Inform students about topic distribution and act as a consolidated archive for exam papers. Provide additional meta data over the existing Crypt exam vaults.

## How?

Enables labelling of question papers.

## When?

Mostly required during exam times.

## For Who?

For students who are facing their exam. Acts as a prioritizer for their exams.

## Design

A dashboard that invokes the metaphor mentioned. Cross link to the universe metaphor.

Cumulate all notifications then filter them based on the tag. Does a tree based navigation make sense here?

## 5.1 Evaluation

No theory is complete without proof that it works in practice.  To demonstrate that these ideas get at where I was aiming, I present a number of case studies I have conducted with the products I have described so far.

Computer technology changes quickly and as a result specific technical knowledge, though useful today becomes outdated quickly. Seen from this lens the following were the things that this dissertation provided me of value.

Conway's Law Conway's law is an adage named after computer programmer Melvin Conway, who introduced the idea in 1968; it was first dubbed Conway's law by participants at the 1968 National Symposium on Modular Programming. It states that

organizations which design systems … are constrained to produce designs which are copies of the communication structures of these organizations

—M. Conway

## 2.5.4 Tradeoffs of Open Source

Things are open. This level of transparency lets the designers to create their own systems and include the necessary changes in the other apps to reflect this. This means that interoperable technologies and other systems that fit into the ecosystem can be introduced with much ease (Invoke the metaphor of LEGO here?)

## 2.5.5 Developer Accounts Future

Looking towards the future of this ecosystem, one of the things that I think would make sense is to have an . It can be administered from the documentation websites [cross-link] that have been mentioned previously.

## 2.5.6 Design Process

Once systems are divided into relatively independent parts, the third step of the design process occurs: creating visual representations as fit solutions for each sub-part. These sub-parts are then visually fitted together into a whole.

## Time Log

Include in Appendix.

## Path Followed visualization

## Reliability

Single point of failure. A way to improve this can be found in the future section.

Current systems without fail would have bugs when corner cases arises. Even though I have tried to prevent them by writing tests, it only works to the effect of checking only for the problems that I could forsee. In another words, it is limited by my understanding of the problem. But the corner cases could be much larger than that.

If time allows, I would opt for test.check, a generative testing library that opts for far more coverage than traditional tests. But even then I think a better idea would be to create fault tolerant programs and this indicate that I have to allow for some resilience. The amount of resilience in the current software is very minimal to non-existent. If something goes wrong, some manual recovery from my part would be necessary to address these problems. This have to be eradicated and as I point out in the [link here] future section. This can possibly be addressed by adopting an Erlang like fault tolerant behaviour.

Clojure enabled EDN over the wire.

## What this project endowed me with

The decision to use functional programming has been great to appreciate the fact that there is more than one way to reach your destination. And as Alan Kay writes in Computer Software in Scientific American, the intellectual limits of Computers are not yet understood and this has been an experience which more emphasized the need to understand clay so as to construct better pots. Computers are capable of self-interpretation whose upper ceiling has not yet been understood.

## Growing a community

Every complex system is made alive not by the code but by the action of people who use the system to their ends. Starts from March 1 with Bundles and ModuleBridge. The toughest test of the system is when it tries to be useful in contexts where the designer didn't indent it to work. It hence was an interesting experiment to see what kind of uses people liked to put these systems into.

## The Right Kind of Pessimism

The right kind of pessimism is one where you assess the problems even after something works. It is good to see these systems as good looking or may be even well performing but one of the core values underlying in the construction of such a system is to focus on the kinds of things that this disables and a few such views. But this is really limited as it's the creator evaluating these systems and it is any way going to be an attempt in futility because I'm behind the creation and there's only so much that I can see. I will leave it to the reader to assess the system, break it down and take away the components if any from the source code (if you can make your way through those lisp paranthesis first)

## Time Tracking

While tackling the project it is easy to miss the forest for the trees and the trees for the forest but it is even tougher to miss the incumbent drought that is about to hit the forest. I have exercised my utmost efforts at crafting the apps here, but for someone at my level of expertise can’t possible handle the extended usage and adaption of the ecosystem, constant pruning of the systems and subsystems is to be maintained if the ecosystem is to be prolonged.

## Unforeseen usages

Perhaps the best test of a system is how it acts when users try to achieve new things with this. Being the designer of this system I am illposed to judge this but I think it might be helpful to get information from others after having used this system for a reasonable amount of time.

## Design Stack

Visualize.

## Development Stack

Visualze.

## Future

If the use of the system is to be continued it has to be ensured that high precision performance tuning is only done after properly understanding the problem domain. Even if all the unit tests passes it doesn't guarantee that some of the combination of the functions and certain characterists which can never be unit tested. It is unreasonable to unit test everything. New changes introduce new pathways to failure.

These new forms of failure are difficult to see before the fact; attention is paid mostly to the putative beneficial characteristics of the changes. Because these new, high

## 5.2 Results

The results were mostly positive. And a lot of criticism on the way things have been put together. But as a proof of concept idea and foundational structure the results prove a much better foundation than the current one. I think it owes a large share to the design language which played a unifying language. Reduce this down to questions to fill in after evaluation.

## Room for focused improvement

A lot of room has now been generated to focus on the improvement.

## Meeting the end goals

The system as it stands now is a pretty barebones

## 5.3 Learnings

One of the main take aways from this project was to develop the app in an iterative loop rather than an incremental fashion. A lot of times I ended up painting the picture in an incremental fashion where in I would complete one app of the ecosystem. Premature optimization is the root of all evil. Even though this was in my mind when constructing the app. I lost the forest for the trees. I was making sure that I don't optimizing the function. But what I had to do was to take in view the entire system complete one whole loop and then start painting in the details. This is something that I would be using throughout my life from now on.

## Incremental vs. Iterative

One of the major learnings that I have made in the development of software was that development is better done in loops of iterative improvements than piecemeal incremental stuff. Lack of adoption of this guiding principle turned out to be detrimental even near the end of writing this dissertation when a certain bug in the PDF exporting system caused lose of considerable amount of time. Developing one complete loop and then strengthening it with additional support structures is what I think a superior way to develop software is rather than putting it together incrementally.

## 6.1 Looking Back

Looking back at this project I think it has been a big journey trying to figure out a lot of Clojure. Learning some pretty life changing concetps from functional programming lanuages.

## 6.2 Future

What the future holds.

### 6.2.1 Next Apps

The current ecosystem could really benefit from

### 6.2.1 Design Improvements

What advantages this brings in from a tutor's perspective and university's perspective.

Learning data from the students, mockup a rough UI mentioning how a tutor could employ and learn from the data accumulated.

And cross department functionality.

### 6.2.1 Code abstraction improvements

### 6.2.1 Performance improvements

### Raise the abstraction bar

I see if further work is to be carried out, the ante of abstracting the program creation can be further elevated and reaching a position where components can be put together by composing higher order functions.

### 6.2.2 Distributed Architecture

Central system means that single point of failure.  I have been looking at OTP architecture adopted as the core programming model for distributed programming in Erlang and it looks like a promising model to adopt for this product.

It would be an interesting problem to see how the data coordination which is much simpler now owing to a central consistent source is to be maintained across multiple systems with unified knowledge representations.

## OTP Architecture

Soft real time system. Bring in Erlang.

## 6.2.3 Administrator/Tutor frontends

This project cannot be deemed to a complete system without having a complementary tutor frontend. While the project didn't give me room for creating a administrator frontend. It was also out of the scope because of the given time frame.

## 6.2.4 Multiverses

## 6.3 Bibliography

-
- [Ohno 1988] Toyota Production System: Beyond Large-Scale Production, Productivity Press, ISBN 978-0-915299-14-0
-
- [Lakoff Johnson 1980] Metaphors We Live By, George Lakoff and Mark Johnson
-
- [Kay 1984] Computer Software, Alan Kay, September 1984. Accessible at:
- http://www.vpri.org/pdf/tr1984001_comp_soft.pdf
-
-
- [Rosch 1978] Rosch, E. (1978). Principles of categorization.
-
- [Shklar Rosen 2009] Web Application Architecture: Principles, Protocols and Practice (2nd edition) Leon Shklar and Rich Rosen
-
- [Nielsen 1995] Jakob Nielsen Usability Heuristics. Accessible at:
- http://www.nngroup.com/articles/ten-usability-heuristics/
-
-
- [ISO9421] ISO9421. Accessible at:
- http://www.iso.org/iso/catalogue_detail.htm?csnumber=16883
-
-
- [Schiff 2015] Humanist Interface by Eli Schiff. Accessible at:
- http://www.elischiff.com/blog/2015/2/2/humanistintroduction
- [Apple 2002] Apple Aqua Human Interface Guidelines
-
- [Allan 2012] The Flat Design Era
- https://web.archive.org/web/20150429184523/http://layervault.tumblr.com/post/32267022219/flat-interface-design
-
-
- [Raskin 2002] The Humane Interface, Jef Raskin, ISBN 0-201-37937-6
-
- [Cook 1998] How Complex Systems Fail, Richard Cook. Accessible at:
- http://web.mit.edu/2.75/resources/random/How%20Complex%20Systems%20Fail.pdf
-
-
- [Knuth 1992] Literate Programming. Donald E. Knuth. University of Chicago Press. ISBN 0-937073-80-6
-
- [NCLMain] NCLVerse Main Website:
- http://nclverse.github.io/
-

-
- [NCLDocs] NCLVerse Documentation Site:
- http://nclverse.github.io/docs
-
-
- [NCLCode] NCLVerse Code Site:
- http://nclverse.github.io/code
-
-
- [NCLDesign] NCLVerse Design Site:
- http://nclverse.github.io/design
-
-
- [NCLDissertation] NCLVerse Dissertation:
- http://nclverse.github.io/dissertation
-
-
- [NCLDocsGen] NCLVerse Documentation Generator:
- http://nclverse.github.io/code/docs-gen/
-
-
- [Transit 2015] Transit Accessible at:
- https://github.com/cognitect/transit-format
-
-

## Early History of Smalltalk

## Erlang Documentation

http://www.erlang.org/doc/design_principles/des_princ.html

## Systemantics

http://www.amazon.com/Systemantics-Systems-Work-Especially-They/dp/0812906748

## Growing a Language

http://www.cs.virginia.edu/~evans/cs655/readings/steele.pdf

## Technologies Used

## Programming Language

Clojure

## Clojure Libraries Used

Ring, Compojure, Enlive, Hiccup, Garden, Om, Clygments, Stasis, Clj-Pdf, Prone, Liberator, Ragtime, Clj-http, Buddy, Yesql, Environ

## IDEs

Emacs LightTable

## Version Control

Git

## Other Technologies

HTML, CSS, Javascript, Postgres, Lentic