# 1 Introduction

This sub-report documents the efforts made by myself in researching appropriate material for the design and development of the final deliverable. To show the full extent of the research performed, all of the stages involved have been included, even aspects that won’t make it into the final design of the end deliverable.

The purpose of this report is to not only to show what research has been carried out, but to also show how the research will ultimately affect the end deliverable. The main contents of this report will discuss:

* Software Evaluation
* Platforms and technologies
* Methodologies
* Academic research
* Evaluation of commercial alternatives

The project I have undertaken is fairly technical in nature, therefore, I have split the report into themes that are common in computer science and other relevant areas.

## Research

//Literature review..//

//Experimentation//

//..,. in the worst case that there is no deliverable, I have made the assumption that I will be evaluating my system myself, using methods that will be researched at a later date…//

# 3 Project Planning

At the beginning of the project I detailed all of the tasks that needed to be completed in a gantt chart. I placed all of the activities that needed to completed in a sequential fashion, where one task led to the next. Research unveiled that what I had implemented was the waterfall process model. Hughes and Cotterell (2006, pp.//FIND PAGES//) explain how students can easily misjudge time allocation of tasks, simply due to the fact that they will be using tools that they are unfamiliar with. This point will play a major factor in my planning approach as most of the tools/software/techniques I will have never used.

//diagram of Gantt chart//

## Waterfall approach

The waterfall method is seen as one of the more basic models of system development. It typically follows the pattern as shown below.

As Hughes and Cotterall (2006, pp. 75-76) explain, the waterfalls main strength is also its downfall; its lack of flexibility. The clear-cut nature of the waterfall model promotes strict adherence to deadlines, however there is great importance in defining the exact deliverables early on as there is very little room for change later on in the development process.

For very straightforward projects with low risk of change, the sequential nature of the waterfall model is ideal.

//pros, small summary//

//cons, why it isn’t suitable for my project//

## Incremental Approach

With the limitations of a //waterfall model/sequential// model in mind, I searched for a process model more suited to this type of project. The spiral model is an iterative process model. As Hughes and Cotterall (2006, pp. 76-77) explains – Aspects of a system that are hard to specify could be better developed using an incremental approach. The spiral model’s key strength is the fact that it is able to cope if various phases of the project encounter complications, or unforeseen issues that hinder development.

Incremental development is well suited to experimentation and prototyping; Sommerville (2010, pp. 32-31) explain that incremental development achieves a final system implementation by progressively validating new versions. As this is such an early stage in my project it is impossible to pin down a time allocation for every single task that I will be performing. If I had greater experience in risk assessment, or the system was very clear to produce then maybe a waterfall model would be the better choice to base my system upon. The fact that an incremental approach, such as the spiral method, accommodates possible contingencies in development, this makes it an ideal approach to use.

//show example spiral process/ or my own//

An iterative approach looks almost perfect, so what are the disadvantages? Is there a limit to the amount of iterations a system can cycle through? Sommerville (2010, pp. 34-35) explains that an incremental approach may degrade a system’s structure, as functionality is constantly being ‘tacked on’, and resources can be used up by constantly documenting new implementations.

## Summary

There are a lot of factors to consider when implementing a project plan. Every process has its pros and cons, however there is flexibility when it comes to choosing. A process model is simply an abstraction for a certain application, which means it doesn’t have to be strictly adhered to. The most advantageous approach for this project is most likely incremental development. With the amount of uncertainty within this project, the most valuable asset I need in a plan is flexibility. Iterative development excels when it comes to trying out various features through prototyping – a task that will feature heavily during the development of the system.

I believe that the disadvantages of an incremental approach won’t affect my development in any serious way. My project is small compared to the large projects that these process models have been based upon. This point alone should lessen the impact of any disadvantages. Documentation will be an essential part of my project, and I doubt my small system will go through enough iterations for it to become unstable.

# 2 System Overview

My main objective is to develop a working prototype of the system. As stated above, the focus at this current time is to develop server-side environment that provides collaborative functionality to multiple client devices. This section discusses the purpose of the system and what the system’s functionality is. This is by no means the final layout of the system as continued research and experimentation continue to show other avenues of development.

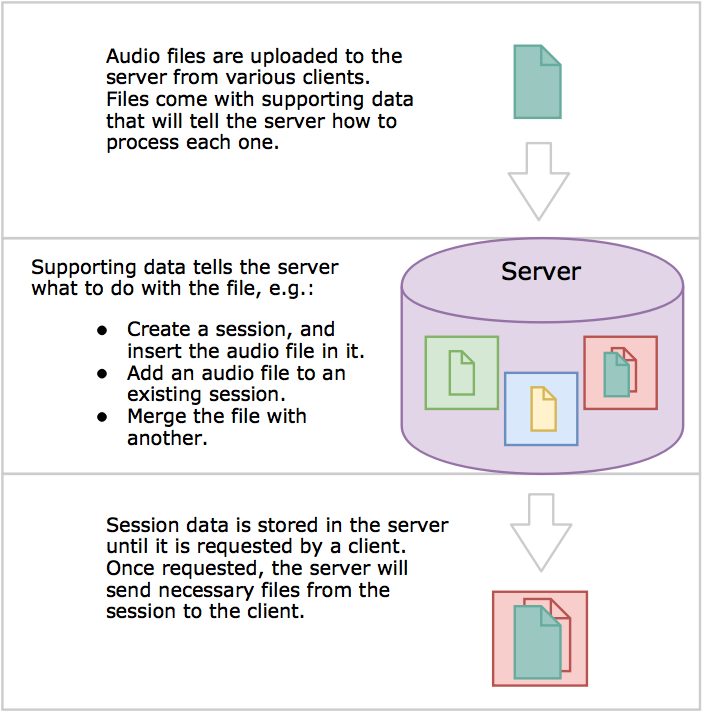
The system requirements haven’t changed drastically during my research, apart from the focus changing to a server-side system. My research was based off of fulfilling the requirements outlined below.

## 2.1. Initial Proposal

The first proposal for the system specified the development of an environment that would allow users to upload and interact with each other’s content via a //remote// server. The initial proposal was heavily centred on a client/mobile device, with the mentality that the GUI and //network programming// would be the main technical hurdle to overcome. However as I continued my research, I found the opposite to be true. My research unearthed that there are many //frameworks/open source aids// that can create a basic client application for you on many different platforms with ease. I found that servers on the other hand, or at least frameworks for collaborative audio specifically, were virtually non-existent. For this reason the main focus was shifted to researching how the server will operate.

### Initial Server Setup Diagram

This diagram was created at a very early in my research stage to better define what was being created. It shows, very basically, what a client will typically upload/download and how the server will process these files.



### Use Case

Below is a very basic use case diagram outlining the interactions of the system with its environment. I used this diagram to help construct my system requirements.

//Use case diagram//

## 2.2. Basic System Requirements

The system requirements haven’t changed drastically during my research, apart from the focus changing to a server-side system. My research was based off of fulfilling the requirements outlined below.

### 2.2.1. Functional requirements

Functional requirements are determined by the outcome of individual functionalities within the system itself. From an early point it was established what actions the system would need to perform. Functional requirements are typically generated from a use case diagram The functional requirements of my system are as follows:

### 2.2.2. Non-functional requirements

A non-functional requirement can be outlined as a characteristic that affects the broader elements of the system. Sommerville (2010, pp. 87-91) states that the non-functional aspects of a system are often the most critical. A user can typically avoid an encounter with abnormal system functionality, whereas a single non-functional feature that isn’t meeting requirements can often cripple the entire system. The non-functional requirements that affect my system include:

#### Implementation

As the system currently stands, all programming for the server will be carried out with the Python programming language, and will be executed on a Linux-based platform. The client prototype will be implemented using Python’s TkInter GUI library. If the project is then ahead of schedule then a final client for mobile will be developed using Intel’s XDK platform using JavaScript, HTML5 and CSS.

#### Performance

The nature of the system will mean that the server will have significant overhead, especially if there are many clients interacting with the server at once. This will mean techniques, such as concurrent programming, may need to be employed to achieve a more efficient system. //Users could potentially be equally distributed in a multithreaded system.//

The client requires slightly less computation, however, audio file formats will need to be used efficiently in order to get quick transmission between client and server.

# 4 Existing Technologies

The initial steps of my research involved finding similar systems that already existed. This research uncovered solutions to certain implementation issues.

## Collaborative Services

### Kompoz

Kompoz (//ref//) is

### Splice

Splice (//ref//) could be defined as collaboration as a service. Its essentially a cloud service that allows users to upload and download each others projects. The main target audience seems to be musicians creating mainly electronic dance music, however it would seem there is nothing stopping other genres from using the service.

A Splice client must be downloaded to the user’s computer in order to upload and download projects. Projects must be made in a compatible digital audio workstations (DAW’s), such as Ableton Live 8 or Logic Pro X //refs?//. The service’s main feature is providing access to other user’s projects, allowing you to create alterations via your personal DAW. Once a user has made adjustments to the project and uploads it, it becomes a new ‘version’ alongside the original project.

In this case no merging of files actually happen unless you consider the operations that the user performs in their DAW. This software highlights a different approach to collaboration: all users can create their own versions, without a final ‘merging’ phase.

### Jamly

## Collaborative Tools

..Due to the collaborative nature of this system, I looked for collaborative services or open-source frameworks that may provide some insight to how to deal with multiple users uploading/editing shared content. //I found

### GitHub

## Technical Audio Tools

### Audacity

..I looked for open-source.. ..because..

..viewed code in xCode.. ..why?.. ..it showed..

## Summary

### Collaboration

As seen above, collaboration can be implemented using many different techniques. In terms of collaborative functionality between users, two patterns seem to be emerging in all of these services and tools:

#### Structured

There is typically a hierarchical structure of users that govern how files/content is authorised into a final/master ‘mix’. All of the collaborative services keep the collaboration process ‘on rails’, a linear approach, however one that could be deemed suitable for its application.

…Github //and// on the other hand gives users full reign of what they can change and upload, however, if conflicting files have been made, or multiple users try to upload to the same branch there can be great complications when trying to merge it back into the original ‘Master branch’.

#### Non-Structured

# System Development

## Client Development

Web vs Native vs Hybrid

XDK

Xcode

## Server Development