LOUGHBOROUGH UNIVERSITY

ATOS IT CHALLENGE

Project Initiation Document

Team Hydra:
Oliver WOODINGS
Jay VAGHARIA
Simon KERR

October/November 2014

1 Introduction

2 Project Plan

3 Project Management

4 Ideas

5 Literature Review Outline

6 Software Development Approach

Software Development Approaches have existed since the 1970s. They range from rigid, long-term methods such as *Waterfall.* through to experimental and risky approaches like *Extreme Programming.* Factors such as team size, business goals and clients all contribute to choosing the correct approach.

To identify the correct software development process to use we must first decide on some attributes of the project of which we can compare across various approaches. The following have been derived from the project ideas and plan:

1. **Team Size** - Most of the development throughout this project will be done by one person. This removes the need for large-scale approaches such as *Waterfall* and *Spiral* and instead opens up the possibility of a faster, looser process such as *Agile.*

2. **Client Interaction** - If the project idea is accepted we will be allowed to work very closely with a representive from Atos. This means our development approach needs to incorporate this throughout the entire lifecycle, making methodologies such as *Waterfall* less relevant since they do not encourage client interaction during the development stage.

3. **Timeline** - The entire development and testing cycle for the project is only a few months, so whatever approach is chosen needs to be dynamic and allow for quick iterations without the need to go through the entire process again. This requirement makes approaches such as *Incremental Development* inappropriate because they add in repeated requirements analysis and architecture design steps that may not be necessary.

4. **Deliverable** - The client is expecting a working prototype to be delivered at the end of the project. The technology must be built to a satisfactory standard, but does not need to be production-ready nor be immaculately written. This allows for the testing, integration and implementation steps in the software development approach to be removed as a priority and opens up the possibility of using less formal approaches such as *Prototyping* and *Extreme Programming.*

These requirements narrow the scope down to three relevant approaches:

- **Agile** - Agile development promotes principles such as continous improvement, early delivery, adaptive planning and evolutionary development. It breaks tasks down into small increments, with the goal of each increment being a new release. There is a big focus on quick communication and easy feedback, normally achieved by daily standup meetings and regular planning sessions. Whilst the communication principles would be useful for this project, the approach also tends to have a relience on multiple teams tackling the project which is not relevant.

- **Prototyping** - Software prototyping is perfect for projects where a production-ready deliverable is not required. Each iteration of the project has no requirement to fully work or meet the users' requirements. Instead, the software goes through an iterative modification process until user acceptance is met or the prototype is discarded. The downfall of prototyping is that it is a very relaxed approach. Client interaction is suggested, not enforced, and often the developers may lose sight of the completed project by getting too involved in a limited prototype which ultimately is only a partial representation of the desired product.

- **Rapid Application Development (RAD)** - RAD is used to describe alternatives to the *Waterfall* approach that focus on iterative development rather than formal, structured planning cycles. The most common definition of RAD, created by James Martin, involves an initial requirements planning stage followed by an interative construction and user feedback stage.

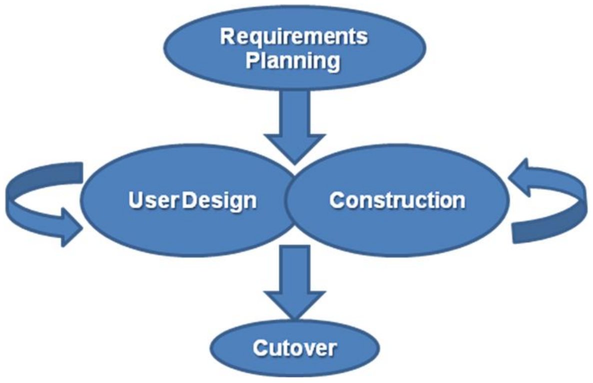

Figure 1: Phases in the James Martin approach to RAD

This fits in nicely with the four project attributes we defined above:

1. **Team Size** - RAD imposes no requirements on number, size or iteractions of teams.
2. **Client Interaction** - The UserDesign stage requires development to go hand in hand with client feedback and testing.

3. **Timeline** There is no restriction on the length of the UserDesign/Construction process in RAD, allowing for the development cycle to last as long as possible before the deadline is reached without having to estimate iterations during early planning stages.
4. **Deliverable** There are no formal requirements for repetitive testing and quality control. This can be implemented independently from the software development approach, allowing for testing to be suited to the individual project.

The analysis above clearly identifies *Rapid Application Development* as the most appropriate software development approach for this project.

7 Technical Considerations

The technical considerations for this project can be divided into three sections: mobile application, server-side systems and hardware.

7.1 Mobile Application

One of the basic top-level requirements for this project is that the end product is centered around a mobile application. Therefore we have to ensure that all technology used is compatible with as many mobile devices as possible. If the app starts to have a dependency on niche features, such as NFC and 4G, there is a risk of excluding large quantities of users.

- **Platform** - There are currently three major mobile platforms that support apps: Android, iOS and Windows Phone. Others such as Blackberry and Firefox OS either do not have a thriving application community or have a very small userbase. This project should ideally support all three of these major platforms, however each one requires applications to be written in different languages. There are two options here which will need to be considered:
  - *Native Applications* - A native application is one that is written to work directly with the device's operating system. For example on Android this would mean that the application is written in the programming language Java and that it directly uses the device's native libraries. The advantage this approach is mainly performance, but there are also certain features that are only available for native applications. The disadvantage is that the application has to be written separately for each platform you want to support.
  - *Framework Applications* - To solve the difficulties of supporting multiple platforms, frameworks such as Apache Cordova and Adobe Phonegap have been created. These allow developers to write their applications once and have it automatically work across all major platforms. The downside here is that the application normally exists inside a ẃrapperẃhich can have a negative impact on performance.

- **Technology** - Features such as NFC are starting to become prevalent in many high-end smartphones, however they have not penetrated the lower-end of the market. The project needs to make sure that proper analysis is done on the prevalance of these features if they are to be used.

7.2 Server-Side Systems

Many ideas and concepts in this project revolve around connected data and technology. There are likely to be many different services and systems required in order to complete the project. Designing and architecturing will be an important part of ensuring the project is a success.

- **Architecture** - There are many different styles of software architecture. For this project we will be using the *Microservices Architecture*. This is centered around separating out the application into individual services, each one with its own unique responsibility. This fits in nicely with the chosen Software Development Approach - *Rapid Application Development* - since it allows for entire services to be completed in a single development cycle.

  Advantages

  - Easier debugging
  - Better fault isolation
  - Independent service development and deployment
  - Removes commitment to a specific technolog stack
  - Avoids monolithic applications and systems

  Disadvantages

  - Requires inter-service communication
  - Testing can be more complicated
  - Multi-service deployment can be hard to orchestrate
  - Multiple services means more memory usage

- **Infrastructure** A solid infrastructure is essential in order to adequately support the numerous software services and systems required for the project. There are several options available when it comes to selecting infrastructure components (servers, database management systems etc):

  - **Cloud Computing** - Cloud-based systems such as *Amazon Web Services* (AWS) and *IBM SoftLayer* allow systems engineers to easily configure and deploy scalable services across the globe. They often support many different types of services (application servers, databases, data processors, queues etc) and can be configured quickly and easily through user interfaces, rather than having to manually set up each service. The downside to cloud-based infrastructure is usually the cost and also the added abstraction; the systems engineer is no longer in direct control of each moving part which can sometimes make fault finding a more laborious process.

– **Dedicated Servers** - Dedicated servers are the more traditional approach to supporting software systems. They give the systems engineer absolute control over the entire infrastructure, however this goes hand-in-hand with the additional responsibility involved in configuring everything by hand. Another concern is redudancy. In a cloud-based system it is very easy to spin up copies of a service in the event of failture, however with a dedicated system you normally need to have servers on standby all the time. Compared to cloud services, dedicated servers can often work out cheaper.

7.3 Hardware

There is the potential for some hardware to be developed and made available to enhance the mobile application. For prototyping it is likely that pre-existing devices such as the Raspberry Pi will be used. Since the main goal of the project revolves around the app, it is unlikely that much time or development will be put into the hardware itself and focus will instead be given to the software that it runs.

8 Risk Assessment