COMS3002 - Software Engineering

Lab 2: Software Requirement Specification

Aobakwe Mosweu (1049090),
Dario Vieira (806414),
Seale Rapolai (1098005),
Wade Downton (1171712)

August 28, 2017

Contents

1 software development life-cycle (Dario Vieira)

1.1 The 6 most common SDLC models

- **The Waterfall model** is a model that does not allow for the revision of a previous phase in the cycle. This model will not be used as a delay in one phase will have a domino effect on all following phases and therefor one member that does not work effectively will hold back the others.

- **The V shaped model** is an adaptation of waterfall that has testing done at the end of each phase to prevent errors/discrepancies before moving on to the following phase. This model will not be used for the same reason as the waterfall model.

- **The Iterative model** allows for a quick production of a single version on a certain set of requirements and allows for a simple means to implement changes.

- **The Spiral model** is an adaptation of the iterative model that will run multiple cycles before the project is complete. Suffers the drawback that it may be drawn into an infinite spiral.

- **The Big bang model** is a haphazard methodology that no planning is done. This is normally used when the requirements are not completely understood by the client or the developer. This will not be used as it is only usable in small groups of programmers (1 or 2) and on small scale projects.

- **The Agile model** is a further adaptation onto the Iterative model that still produces a product very quickly but involves the client and user more regularly to ensure it is meeting the expectations. The main drawback to Agile is because of the user centered approach the user may lead the project astray if they are not clear.

We will be using an agile approach as it is the current industry standard and we believe it will allow us to bring the most effective product in the end, especially considering on the wide variety of people that out project will be appealing to.

2 Application Architecture (Wade Downton)

2.1 Types of Architecture

2.1.1 Blackboard

A blackboard system is an artificial intelligence approach based on the blackboard architectural model. The architecture relies primarily on machine learning and AI to solve complex, NP-Hard problems.

Pros:

- Made to handle complex, ill-defined problems with the use of AI.
- Is able to easily adapt to new data.

Cons:

- This architecture does not scale well to real world problems.
- Very complex to design and implement.
- Requires a large amount of pre existing data to train algorithms.

2.1.2 3-tier Client-Server

A distributed application structure that partitions tasks between the providers of a resource or service. A three-tier client/server is a type of multi-tier computing architecture in which an entire application is distributed across three different computing layers or tiers. It divides the presentation, application logic and data processing layers across client and server devices. The client will request from the application server and the application server will request from the database server. The client will never communicate without the use of the database server.

Pros:

- All files are stored in a central location.
- Backups and network security is controlled centrally, as only the application server can access the database server.
- Users can access shared data which is centrally controlled.

- The workload on the client is lowered with the use of a server.

Cons:

- The server are expensive to purchase and maintain.
- If any part of the server side network fails then a lot of disruption can occur

2.1.3 Monolithic application

A monolithic application is built as a single unit. The application is not separated into smaller partitions and services. The single unit involves the program and its own data storage. No centralization exists in this architecture without the use of a server.

Pros:

- Simple to design and create.

Cons:

- Designed without modularity, i.e. Cannot add to or alter the application without difficulty.
- Does not scale well for larger applications.
- Can be difficult to debug, as problems can be difficult to locate on large scale applications.

2.1.4 Peer-to-peer

An architecture that partitions tasks or workloads between peers or nodes, where no centralized node exists. Data is shared amongst all the nodes and it is easy to remove or insert nodes into the system without affecting the system.

Pros:

- More reliable as central dependency is eliminated.
- The over-all cost of building and maintaining this architecture is comparatively very less than client-server.

Cons:

- Weaker security over data than that of client-server.
- Data recovery or backup is very difficult.
- Difficult to update applications over this type of architecture.

2.1.5 Rule-based Architecture

The use of forward and backward chaining in order to determine rules that define the architecture of a system. Forward chaining observes the given data and makes decisions based on that. Backward chaining observes the goals and uses that to make decisions.

Pros:

- Easy to implement by defining separate rules in separate layers.
- Allows for modularity.
- All "knowledge" in the architecture is uniform as it can be expressed in the same format defined by the rules.

Cons:

- Limited amount of rules that can be defined.
- Complexity increases as the system size increases.

2.2 Choice of Architecture: 3-tier Client-server Architecture

This architecture provides for an easy, separate development of the back-end and front-end of the application. Data will be centralized on the server that will be used and this will provide for strong layer of user privacy and security. Modularity is easy as alterations to the application can be rolled out on the server side towards all clients at once. This architecture is best for this scale of application, and a lot of processing will be done server side so less workload will be put on the client.

3 Front-end User Interface (Aobakwe Mosweu)

3.0.1 Front-end User Interface Methods

1. Website: Browser
2. Mobile App: Android and other platforms
3. Desktop App

3.0.2 Choice of User-Interface

I think the best UI to use for our project is a website, because it will be accessible on any device that has a browser and an internet connection. It will compatible with most devices i.e.) It will not be restricted to only a computer or a phone.

Websites are accessible to a widest possible audience, as compared to applications that will still have to be downloaded. The maintainer can instantly add some updates to the website and users dont have to download the latest version of the website to be able to use it.

Websites can easily be shared between users by a simple link over social media, whereas with applications it is not that easy to do so.

Maintenance and support of websites are not that expensive whereas with applications you must do upgrades, testing and take care of compatibility issues, which is much more expensive.

Websites are much easier for users to find because their pages can be displayed in search results.

4 Back-end Service and DBMS (Seale Rapolai)

4.1 Back-end Service: NodeJS

NodeJS is a platform build using Javascript. It is easy to learn and allows developers to build fast and scalable network applications. It uses an event driven, non-blocking I/O model that makes it lightweight and very efficient. This makes it suited for data-intensive and real-time applications that have have high network traffic. It is open source and cross platform, which makes it very portable and versatile. It also has a vast module library which simplifies the development of applications.

Advantages:

- **Asynchronous and Event Driven** - NodeJs APIs are asynchronous (non-blocking). This means the server never waits for API to return data before it has to handle other requests.
- **Fast** - It was build on Google Chrome's V8 JavaScript Engine and as such has fast code execution.
- **Single Threaded but Highly Scalable** - It is sigle threaded with event looping. This helps the server to respond in a non-blocking way since each request is handled on after the other. This makes server very scalable as compared to other servers that have limited number of threads to handle requests.
- **No Buffering** - A NodeJS server never buffers data, but instead outputs data in chunks.

4.2 DBMS: MongoDB

MongoDB is a NoSQL database developed to deal with the limitations of SQL databases which include scalability, multi-structured data, goe-distribution and agile developments sprints. It stores data in flexible, JSON-like documents where fields can vary from document to document. This implies that the data structure can change over time. The document model maps to objects in application code and makes data easy to work with. MongoDB uses Ad hoc queries, indexing and real-time aggregation to provide powerful ways to access and analyse data and it is free and open source.

5 Supporting Software (Dario Vieira)

5.1 Google Places API

This will be used to locate all places of interest in a certain area such as hospitals, schools, airports, etc. This will be necessary as producing a list of all the places on interest accurately is not viable.