COS301: Meeeting Notes 3 March, 2015

# Introduction

Many definitions of software architecture exists, as can be read about in the pdf on the CS website. A more general definition (as provided by Microsoft) is as follows:

“ Software application architecture is the process of defining a structured solution that meets all of the technical and operational requirements, while optimizing common quality attributes such as performance, security, and manageability. It involves a series of decisions based on a wide range of factors, and each of these decisions can have considerable impact on the quality, performance, maintainability, and overall success of the application. “

Thus software architecture is about making decisions related to how and with what a software system will be constructed.

# Our requirements for Phase 2

There are 4 sections that we have to complete for phase 2 of the mini project. These are:

* Access channel requirements
* Quality requirements
* Integration Requirements
* Architecture constraints

## Access Channel Requirements

This does not only include the various platforms (android, iOS etc) that our system will work on, but also the networking channels that will be used to accomplish communication between the software devices and the various endpoints. A few examples follow:

### Restful web services:

*Representational State Transfer (REST) is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modifiability, that enable services to work best on the Web.*

Read more here: <http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html>

### SOAP

*SOAP, originally an acronym for* *Simple Object Access protocol, is a* [protocol](http://en.wikipedia.org/wiki/Protocol_(computing)) *specification for exchanging structured information in the implementation of* [web services](http://en.wikipedia.org/wiki/Web_service) *in* [computer networks](http://en.wikipedia.org/wiki/Computer_network)*. It uses* [XML Information Set](http://en.wikipedia.org/wiki/XML_Information_Set) *for its message format, and relies on other*[application layer](http://en.wikipedia.org/wiki/Application_layer) *protocols, most notably* [Hypertext Transfer Protocol](http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol) *(HTTP) or* [Simple Mail Transfer Protocol](http://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol) *(SMTP), for message negotiation and transmission.*

Read more here: <http://en.wikipedia.org/wiki/SOAP>

Many other technologies exist that provide channels of access for software systems.

## Quality Requirements

A good definition of quality requirements is as follows:

*Quality attribute requirements.* These requirements are qualifications of the functional requirements or of the overall product. A qualification of a functional requirement is an item such as how fast the function must be performed, or how resilient it must be to erroneous input. A qualification of the overall product is an item such as the time to deploy the product or a limitation on operational costs.

This definition highlights the important point that the quality requirements must be matched to the functional requirements. As an example a functional requirement of the BuzzSystem is that a user must be able to close a thread. A quality requirement might be that opening and closing of threads must be audited (this refers to the auditability of the system) – **This is just a made up example!**

**Quality requirements must be prioritized, since no system can be fast, secure, flexible, reliable, scalable etc at the same time.**

According to out meeting today, 3 of the most important quality requirements of the BuzzSystem is:

1. Scalability
2. Usability
3. Auditability (for marks)
4. Performance (This was hinted at by Stacey today, she mentioned “if I have to wait 10 minutes to open a thread I won’t want to use the system).

We need to specify how we will achieve these core quality requirements. Here we can refer to various architectural patterns that we can use to achieve these quality requirements. The way architectural patterns achieve quality requirements was discussed in class and is also available in the Architecture Design PDF on the CS website. Have a look at the patterns, especially those relating to the quality requirements mentioned above.

## Integration Requirements

The integration requirements referred to here is not concerning the integration of the various modules of the BuzzSystem. These are the integration requirements for external system. This definitely includes LDAP (see phase 1 memo spec for more information).

We do not need to provide API specification on the form of UML models, but where APIs are used, we will have to briefly describe them and what their purpose is.

Something of interest:

*APIs often come in the form of a*[*library*](http://en.wikipedia.org/wiki/Library_(computing))*that includes specifications for*[*routines*](http://en.wikipedia.org/wiki/Subroutine)*,*[*data structures*](http://en.wikipedia.org/wiki/Data_structure)*,*[*object classes*](http://en.wikipedia.org/wiki/Class_(computer_programming))*, and variables. In other cases, notably*[*SOAP*](http://en.wikipedia.org/wiki/SOAP)*and*[*REST*](http://en.wikipedia.org/wiki/REST)[*services*](http://en.wikipedia.org/wiki/Web_service)*, an API is simply a specification of*[*remote calls*](http://en.wikipedia.org/wiki/Remote_procedure_call)*exposed to the API consumers.*[*[1]*](http://en.wikipedia.org/wiki/Application_programming_interface#cite_note-1)

Furthermore, there might be quality requirements relating to integration itself. As an example, the system integrates with LDAP, and a potential quality requirement in terms of Scalability might be that the channel connecting the BuzzSystem to LDAP must be able to handle the required flow of information, which may increase as more users are added.

## Architecture Constraints

At the moment our client has not specified any constraints to which we must adhere. But there are many architectural constraints that we will have to look at, of which the following are a few:

* Environmental constraints
* Timeframe constraints (We only have a month for implementation!)
* Technology constraints
* Economic constraints (The cost of developing the system)

Our architectural design will probably also bring out various constraints as we continue.

General Notes

* Read through the memo spec. Stacey mentioned that we can use the modules they identified as a way to get ideas for quality requirements. Just remember, the memo spec will change, no doubt about it, nothing is cast in stone. We and the lecturers are together in creating a working system.
* Have a look at the architectural patterns that Java EE employs, and how they can solve some of our quality requirements. Chances are good that we will be using the Java EE reference framework for the backend of our system.