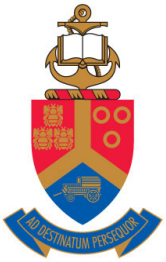

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

COS301 Mini Project Functional Architecture Requirements

Group Name: Group 7_a

Roger Tavares *1*0167324
Thinus Naude *1*3019602
Kabelo Kgwete *1*1247143
Sylvester Mpanganer *1*1241617
Maphuti Setati *1*2310043
Ruth Ojo *1*2042804
Axel Ind *1*2063178
Lindelo Mapumulo *1*2002862
Maria Qumalo *2*9461775

Git repository link:

`https://github.com/thinusn/COS301MiniProjectArchitectureRequirements`

Final Version

March 6, 2015

Contents

1 Introduction

This document was compiled by our group during our meetings and was produced as a whole by the team.

This document contains specifications of the software architecture requirements. This is the infrastructure upon which the application functionality will be developed. The following non-functional requirements are addressed in depth with supporting diagrams (when necessary):

- Access and Integration requirements.
- Architectural responsibilities.
- Quality requirements.
- Architecture constraints as specified by the client.

2 Architecture requirements

2.1 Architectural scope

2.2 Critical quality requirements

2.2.1 Scalability

Description:

Justification:

Mechanism:

1. Strategy:
2. Architectural Pattern:

2.2.2 Security

Description:

Justification:

Mechanism:

1. Strategy:
2. Architectural Pattern:

2.2.3 Usability

Description:

Justification:

Mechanism:

1. Strategy:
2. Architectural Pattern:

2.2.4 Integrability

Description:

Justification:

Mechanism:

1. Strategy:
2. Architectural Pattern:

2.3 Important quality requirements

2.3.1 Performance

Description:

Justification:

Mechanism:

1. Strategy:
2. Architectural Pattern:

2.3.2 Plug-ability(Maintainability)

The system as a whole should be designed and developed in such a way that it is modular allowing for additional modules to be added or even removed.

The reason why this is an important quality requirement is because the system should allow the addition of new plug-ins(modules) or the removal of old modules that are no longer required or obsolete. This allows for a more adaptable system as it can be adjusted to a specific users needs. By ensuring plug-ability the system will also be much easier to test. Diagnostics of potential flaws and module clashes can now be eliminated as the system can be tested as individual modules and even as a whole allowing for multiple module configurations.

Mechanisms:

1. (a) There are multiple strategies that can ensure plug-ability one such strategy as mentioned above is to subdivide the system into separate interconnect-able modules or plug-ins.

   (b) Another strategy is to split the system into many services that communicate with one another to perform the required functional requirements.

2. The best Architectural pattern to realize this requirement would be the micro-kernel, this is because this pattern allows for a plug-and-play infrastructure meaning that modules can easily be added or removed or even rolled back to previous versions without affecting other services on the system. This is done by using the Internal servers of the micro-kernel.

2.3.3 Monitor-ability

Description:

Justification:

Mechanism:

1. Strategy:
2. Architectural Pattern:

2.4 Nice to have quality requirements

2.4.1 Reliability and Availability

Description:

Justification:

Mechanism:

1. Strategy:
2. Architectural Pattern:

2.4.2 Testability

Description:

Justification:

Mechanism:

1. Strategy:
2. Architectural Pattern:

2.5 Integration and access channel requirements

2.6 Architectural constraints

3 Architectural patterns or styles

4 Architectural tactics or strategies

5 Use of reference architectures and frameworks

6 Access and integration channels

7 Technologies