COS 731
Software Engineering II

Lecture 1

Introduction:
An Overview of Software Architecture

20 July 2015
Topics

A. Admin matters
B. Course overview
C. Overview of software architecture
D. Challenges in software architecture
E. Class exercise: Discussions & presentations
A. Admin – general (1)

- Study guide
- Study material / teaching resources
- Course website:  
  http://www.cs.up.ac.za/courses/COS731

- Lecture schedule
- General structure for lectures
- Assessment

- Expectations by the lecturers
- Expectations by the students
A. Admin - Schedule for lectures

For the first 7 weeks

<table>
<thead>
<tr>
<th>Week</th>
<th>Date</th>
<th>Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20 Jul</td>
<td>Course overview &amp; introduction to software architecture</td>
</tr>
<tr>
<td>2</td>
<td>27 Jul</td>
<td>Middleware architectures &amp; technologies (1)</td>
</tr>
<tr>
<td>3</td>
<td>3 Aug</td>
<td>Middleware architectures &amp; technologies (2)</td>
</tr>
<tr>
<td>4</td>
<td>10 Aug</td>
<td><strong>Public holiday</strong></td>
</tr>
<tr>
<td>5</td>
<td>17 Aug</td>
<td>Software architecture process</td>
</tr>
<tr>
<td>6</td>
<td>24 Aug</td>
<td>Documenting a software architecture</td>
</tr>
<tr>
<td>7</td>
<td>31 Aug</td>
<td>Software product lines</td>
</tr>
</tbody>
</table>

**Lecture plan:**

(1) Reading material provided 1 week in advance
(2) student to read papers before the lecture
(3) class exercises in groups (4) selected groups give presentations
# A. Admin - assessment

<table>
<thead>
<tr>
<th>Assessment activity</th>
<th>Category</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class participation:</td>
<td>group</td>
<td>20</td>
</tr>
<tr>
<td>(weekly class presentations)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Theory assignment</td>
<td>individual</td>
<td>20</td>
</tr>
<tr>
<td>Project</td>
<td>group</td>
<td>20</td>
</tr>
<tr>
<td>Examination</td>
<td>individual</td>
<td>40</td>
</tr>
</tbody>
</table>
Definition(s) of Software Architecture

Various definitions exist

**Definition 1** [ANSI/IEEE std 1471 – 2000]

*Software architecture is:* the fundamental organisation of a system, embodied in its **components**, their **relationships** to each other and the environment, and the **principles** governing its design and evolution.

component = a recognisable "chunk" of software
relationships = how components communicate
Definition(s) of Software Architecture

Definition 2 [Bass et. al, 2003]

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.
Software Architecture topics to be covered:

- representation of:
  - designs
  - styles and patterns of software architecture.

- themes include:
  - model-driven architecture
  - formal modelling and analysis
  - architectural description languages.

- significant design decisions concerning:
  - the components that make up a system (application)
  - repeating-patterns of system-wide aspects
  - platforms on which the systems are built.
Software architecture is (definition 1):
the fundamental organisation of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.

So, software architecture:

1. defines structure
2. specifies component communication
3. addresses non-functional requirements
4. is an abstraction
C. Overview of software architecture (2a)

(1) Software architecture defines structure

Partitioning: (a major activity for a software architect)
- Partition an application into a set of inter-related components based on:
  - Application requirements and constraints

Example:

Information management system

- Requirement:
  - E.g. Application is distributed across multiple sites

- Constraint:
  - E.g. Certain functionality & data must reside at each site
C. Overview of software architecture (2b)

Partitioning contnd:

- **assign responsibilities** to each component
  - responsibilities define the tasks to be performed by a component within the application

- **All components** that make up the architecture **collaborate** to provide the required functionality

- **Responsibility-driven design**
  (from OO design) is a good approach to **partitioning**

- **key structural issue:**
  - **minimise dependencies** (=> loosely coupled architecture)
  - a dependency between components A and B
    - => a change in A necessitates a change in B & vice-versa
C. Overview of software architecture (3a)

(2) Software architecture specifies **component communication**

- **Communication**: is in terms of data and control information

- **Architectural patterns (styles)**:
  - structures that facilitate certain kinds of component communication

  - re-usable **architectural blueprints** that describe the structure & interaction between collections of participating components

- **Example**: client-server pattern
(2) Software architecture specifies **component communication** contnd

- **Example: client-server pattern provides**
  - a mechanism for clients to locate servers
  - asynchronous request-reply communication from client to server
  - servers supporting one or more clients through a published interface
  - a mechanism to handle errors
  - (optionally) a mechanism to provide security for server access
C. Overview of software architecture (4a)

(3) Software architecture addresses **non-functional requirements**

- **Functional requirements:**
  - define **what** an application does.

- **Non-functional requirements:**
  - define **how** the application provides the required functionality.

- **3 distinct areas of non-functionality requirements**
  1. technical constraints
  2. business constraints
  3. quality attributes

- Software architect must create a platform to support functional requirements & satisfy non-functional requirements.
C. Overview of software architecture (4b)

(3) Software architecture addresses non-functional requirements contd

3 distinct areas of non-functionality requirements

(i) **technical constraints** constrain design options
- e.g. only Java developers are available, so the application must be developed in Java

(ii) **business constraints** constrain design options
- e.g. in order to widen our customer base we must interface with XYZ tools

(iii) **quality attributes** define applic. requirements in terms of:
- performance, scalability, modifiability, security, availability, portability, usability, testability, supportability, etc.
C. Overview of software architecture (5)

What does a software architect do?

Exact roles & responsibilities depend on environment

(1) Liaison (in order to explain terminology to stakeholders)
   a. between clients and technical team & between engineering teams
   b. with management

(2) Software Engineering
   Excellent design skills enable a software engineer to become an architect.

(3) Understanding and selecting technology
   a. acquire a deep understanding of technology domains
   b. Select appropriate technologies

(4) Risk management
   evaluate risks associated with design choices & technology choices
Class activities (30 minutes)

A. Selection of discussion groups

B. Select an application that you know well (from a user perspective) and answer the following question:

For the following **quality attributes:**

1. **Performance** (throughput & response time)
2. **Scalability** (simultaneous connections & data size)
3. **Availability**
4. **Security**

explain why each quality attribute is important. Refer to the **handout** for the detailed description of each attribute.

**Some examples of applications:**

Facebook  Twitter  YouTube
Online-student-registration  Internet-banking  Load-shedding
Presentations

- Group 1
- Group 2
- Group 3
- Group 4
- Group 5