COS 731
Software Engineering II

Lecture 6
Software Architecture Process (3)

31 August 2015
Topics

Last 2 lecture:

1. Process outline
2. Architecture design: (a) choosing the architecture framework (b) allocating components
3. Validation: (a) Using scenarios (b) prototyping
4. Documenting a software architecture

This lecture:

1. ICDE case study
2. Attribute-Driven design

3. Class exercise: Discussions & presentations
**RECAP: Architecture design**

**Architecture design**

**A. Inputs for architectural design**
- architecture rqmnts documents

**B. Architecture design steps**
1. **choose an overall strategy** for the architecture based on proven architecture patterns.
2. specifying components of the application, how they fit into framework, & allocating component responsibilities

**C. outputs of architectural design**
1. **Views:** capture the design
2. **Documents:** explain design, reasons for design decisions & associated risks
Outputs of architectural design

(1) **Architecture Views**: diagrams that depict the design
- UML diagrams:
  - class diagrams
  - component diagrams, etc.
are a formal way of presenting views

(2) **Documents**: explain design, reasons for design decisions & associated risks
Documenting a Software Architecture (2)

Outputs of architectural design
- Use of a document template e.g.

Project Name: pppppppp

1. Project context

2. Architecture requirements
   2.1 Overview of key objectives
   2.2 Architecture use cases
   2.3 Stakeholder architectural requirements
   2.4 Constraints
   2.5 Non-functional requirements
   2.6 Risks

3. Solution
   3.1 Relevant patterns
   3.2 Architecture overview
   3.3 Structural views
   3.4 Behavioural views
   3.5 Implementation issues

4. Architecture Analysis
   4.1 Scenario Analysis
   4.2 Risks
Attribute-Driven design (ADD)

- Architecture design method developed by the Carnegie Mellon SEI

  - An approach to defining a software architecture in which the design process is based on the software quality attribute requirements

  - Follows a recursive process that decomposes a system or system element by applying **architectural tactics and patterns** that satisfy driving quality attribute requirements
    - **Patterns** are proven solutions to problems. **Tactics** focus on specific quality attributes.
    - **Architectural tactic is a design decision** that affects how well a software architecture addresses a given quality attribute.

  - Follows a “Plan, Do, Check” cycle
    - **Plan:** select types of elements to be used based on quality attributes & design constraints
    - **Do:** Instantiate elements to satisfy quality attribute requirements and functional requirements
    - **Check:** Analyse resulting design to determine if requirements are met.
Gorton:

Architectural Requirements

Choose
Architectural Framework

Allocate Components

Architectural Views

Architecture Document

Functional requirements

Design constraints

Quality attribute requirements

Step 1: Confirm there is sufficient requirements info

Step 2: choose an element of system to decompose

Step 3: Identify candidate architectural drivers

Step 4: choose a design concept that satisfies architectural drivers

Step 5: instantiate architectural elements & allocate responsibilities

Step 6: define interfaces for instantiated elements

Step 7: verify & refine requirements & make them constraints for instantiated elements

Software architecture design
Case study: ICDE (1)

- Information Capture and Dissemination Environment (ICDE)

**Requirements overview:**

- A system to automatically capture & store data on actions performed by a user at a workstation.

- e.g. of data for Google search:
  - search query string
  - copies of pages returned by Google

- Data is to be analysed by third-party tools to provide ‘intelligent’ help to users
Case study: ICDE (2)

Project context

Application architecture for ver 1.0 (single-user, 2-tier)

- Data collection
  - a number of loosely coupled processes
  - transparently track user’s activities & stores them in data store
  - activities relate to Internet access, documents opened & browsed, etc.

- Data store
  - A COTS Relational database system
  - time-stamped data on user activities is stored in several tables

Component diagram for ver 1.0 application architecture (2-tier architecture)

- Data analysis
  - GUI-based tool which supports queries to the DB
  - tool was created for testing purposes
### Business goals: for ver 2.0

<table>
<thead>
<tr>
<th>Business goal</th>
<th>Supporting technical objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encourage 3&lt;sup&gt;rd&lt;/sup&gt; party tool developers</td>
<td>Provide third party tools with simple and reliable access to the data store heterogenous (i.e. not-only MS Windows) platform support for running 3&lt;sup&gt;rd&lt;/sup&gt; party tools</td>
</tr>
<tr>
<td></td>
<td>Allow 3&lt;sup&gt;rd&lt;/sup&gt; party tools to communicate with ICDE users from a remote machine</td>
</tr>
<tr>
<td>promote the ICDE concept to users</td>
<td>Scale the data collection &amp; data store components to support up to 150 users at a single site</td>
</tr>
<tr>
<td></td>
<td>Low-cost deployment for each ICDE user workstation.</td>
</tr>
</tbody>
</table>

### constraints:
- **Time:** interim release after 6 months, final release after 12 months
- **Budget:** fixed budget => influences on: implementation choices, number of developers, etc.
Overview of key objectives for ICDE ver 2.0:

(1) To provide an infrastructure to support a programming interface (API) for 3rd party client tools to access the ICDE data store. Must offer:
   - flexibility in terms of deployment,
   - plug into ICDE environment to obtain or provide info on users activities
   - provide simple read / write access to ICDE data store

(2) Evolve the ICDE architecture to support 100-150 users at low cost
Overview of key objectives for ICDE ver 2.0:

(1) To provide an infrastructure to support a programming interface (API) for 3rd party client tools to access the ICDE data store. Must offer:
   - flexibility in terms of deployment,
   - plug into ICDE environment & obtain or provide info on users
   - provide simple read/write access to ICDE data store

(2) Evolve the ICDE architecture to support 100-150 users at low cost

ADD inputs:

Are these:

Functional requirements?
Design constraints?
Quality attribute requirements?
Architecture use cases for ICDE ver. 2.0 API:

1. ICDE data access:
   - queries from a 3rd party tool focus on a single user. query sequence is:
     - get info. on user’s current assignment,
     - get detail about user activities, analyse activities, create output info for user.

2. Data storage: 3rd party tools need:
   - to store info in ICDE data store in order to share data about analysed user activities
   - to notify users & other tools about availability of new data
   - meta-data on content of data store
Case study: ICDE ver 2.0

Architecture use cases for ICDE ver. 2.0 API:

1. ICDE data access:
   queries from a 3rd party tool focus on a single user query sequence is: get info. on user’s current assignment, get detail about user activities, analyse activities, create output info for user.

2. Data storage:
   3rd party tools need to:
   - store info in ICDE data store in order to share data about activities
   - notify users & other tools about availability of new data
   - to have access to meta-data

ADD inputs:

Are these:
Functional requirements?
Design constraints?
Quality attribute requirements?
2. Architecture requirements

2.1 Overview of key objectives

2.2 Architecture use cases

2.3 Stakeholder architectural requirements

2.4 Constraints

2.5 Non-functional requirements

2.6 Risks

Stakeholder architectural requirements for ICDE ver 2.0:

(1) 3rd party tool producers:
(a) Ease of data access   (b) heterogeneous platform support   (c) Instant event notification

(2) ICDE API programmers:   The API should:
(a) be easy & intuitive to learn   (b) be easy to comprehend & modify code
(c) provide concise programming model for implementing common use cases, etc.

(3) ICDE development team:   Insulate 3rd party tools from the details of, and any changes to the ICDE data store structure, etc..
Stakeholder architectural requirements for ICDE ver 2.0:

(1) 3rd party tool producers:
(a) Ease of data access  
(b) heterogeneous platform support  
(c) Instant event notification

(2) ICDE API programmers:
The API should:
(a) be easy & intuitive to learn  
(b) be easy to comprehend & modify code  
(c) provide concise programming model for implementing common use cases, etc.

(3) ICDE development team:
Insulate 3rd party tools from the details of, and any changes to the ICDE data store structure, etc.

ADD inputs:

Are these:
Functional requirements?
Design constraints?
Quality attribute requirements?
2. Architecture requirements

2.1 Overview of key objectives
2.2 Architecture use cases
2.3 Stakeholder architectural requirements
2.4 Constraints
2.5 Non-functional requirements
2.6 Risks

Constraints for ICDE ver 2.0:
(1) ICDE ver 2.0 database schema must be used.    (2) Must run on MS Windows

Non-functional requirements for ICDE ver 2.0:
(1) Performance: provide < 5 sec response times for retrieval of <= 1000 rows
(2) Reliability: resilience to failures induced by 3rd party tools
(3) Simplicity: => flexible architecture design that is easy to evolve, extend, etc.

Risks for ICDE ver 2.0:
Concrete requirements are not readily available:
Mitigation: keep initial design simple & extensible
Case study: ICDE ver 2.0 (8)

Constraints for ICDE ver 2.0:
(1) ICDE ver 2.0 database schema must be used. (2) Must run on MS Windows

Non-functional requirements for ICDE ver 2.0:
(1) Performance: provide < 5 sec response times for retrieval of ≤ 1000 rows
(2) Reliability: resilience to failures induced by 3rd party tools
(3) Simplicity: => flexible architecture design that is easy to evolve, extend, etc.

Risks for ICDE ver 2.0:
Concrete requirements are not readily available:
Mitigation: keep initial design simple & extensible

ADD inputs:
Are these:
- Functional requirements?
- Design constraints?
- Quality attribute requirements?
3. Solution

3.1 Relevant patterns
3.2 Architecture overview
3.3 Structural views
3.4 Behavioural views
3.5 Implementation issues

Step 2: choose whole system
Step 3: Some candidate architectural drivers:
(a) Ease of data access
(b) Heterogeneous platform support
(c) Instant event notification
(d) Insulate 3rd party tools from the details of data store
(e) Provide < 5 sec response times for retrieval of ≤ 1000 rows
Case study: ICDE (7)

BASED ON DOC TEMPLATE

3. Solution

3.1 Relevant patterns
3.2 Architecture overview
3.3 Structural views
3.4 Behavioural views
3.5 Implementation issues

Step 2: choose whole system

Step 3: Some candidate architectural drivers:

(a) Ease of data access
(b) Heterogeneous platform support
(c) Instant event notification
(d) Insulate 3rd party tools from the details of data store
(e) Provide < 5 sec response times for retrieval of <= 1000 rows

Step 4: choose a design concept that satisfies the architectural drivers

CLASS EXERCISE: Given the patterns discussed so far:

(1) n-tier client-server pattern          (2) messaging pattern
(3) publish-subscribe pattern           (4) broker pattern
(5) process coordinator pattern

What combination of these patterns would be suitable given the above candidate architectural drivers, and why?