Topics

Last lecture:

1. Process outline
   a. determining architectural requirements
   b. prioritising architectural requirements

2. Architecture design: choosing the architecture framework

This lecture:

2. Architecture design: allocating components
3. Validation:
   (a) Using scenarios  (b) prototyping
4. Documenting a software architecture
5. Class exercise: Discussions & presentations
RECAP: Process outline

A 3-step architecture design process

1. **Determine architectural requirements**
   create a statement or model of requirements that will drive the architecture design

2. **Architecture design**
   define the structure and responsibilities of the components

3. **Validation**
   test the architecture against existing and (possible) future requirements
Architectural requirements (1)

**Inputs & outputs** for determining architectural requirements

Architectural requirements
aka
architecturally significant requirements
aka.
architecture use cases

are the non-functional requirements (quality attributes & constraints)
RECAP: Architecture design (1)

Architecture design

A. Inputs for architectural design
   - architecture reqmnts 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

**In this lecture:**
- Allocating components
- Architecture description: views & documents
B. Architecture design steps

(1) choose an overall strategy (to satisfy key requirements) for the architecture based on proven architecture patterns.

(2) allocate components ...........

- 5 commonly used architecture patterns:
  (1) n-tier client-server pattern
  (2) messaging pattern
  (3) publish-subscribe pattern
  (4) broker pattern
  (5) process coordinator pattern

are commonly used because:
  - they are well understood & they work

- Architecture design incorporates one or more patterns depending on complexity of application
B. Architecture design steps

(2) Allocate components:

- specify components of the application
- specify how they fit into framework
- allocate component responsibilities

Framework defines overall communication patterns for components.

Must be augmented by identifying:

- the major application components & how they plug into framework
- the interface or services that each component supports
- responsibilities of each component, e.g. what it must do when it receives a request
- dependencies between components
- partitions in architecture that can be distributed over servers in a network
B. Architecture design steps

Allocate components:

- specify components of the application
- specify how they fit into framework
- allocate component responsibilities

Some guidelines for component design

a. Minimise dependencies between components

b. Design components that encapsulate a highly cohesive set of responsibilities

c. Isolate dependencies on middleware and COTS infrastructure technologies

d. Use decomposition to structure hierarchically

e. Minimise calls between components
2.1 Allocate components:

1. Choose Architectural Framework

2. Allocate Components

2.1 specify components of the application
2.2 specify how they fit into framework
2.3 allocate component responsibilities

structural view of an order processing application

1. Uses **messaging pattern**

2.1 Components are:
   1. OrderInput
   2. Validate
   3. Store
   4. SendEmail

2.2 structural view shows how components fit into framework
(2) Allocate components:

2.1 specify components of the application
2.2 specify how they fit into framework
2.3 allocate component responsibilities

structural view of an order processing application

<table>
<thead>
<tr>
<th>New orders</th>
<th>orderInput</th>
<th>Order Q</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>read</td>
<td>write</td>
</tr>
<tr>
<td>Error log</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>check</td>
<td>write</td>
</tr>
<tr>
<td></td>
<td>order</td>
<td>get</td>
</tr>
<tr>
<td>Validate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Store</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customer System</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Order System</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SendEmail</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Email Server</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

KEY:
Black font: Existing component
Blue font: New component
Red font: Storage
A ----> B: A has a dependency on B

OrderInput
- Accessing new orders database
- Encapsulating order processing logic
- Writing messages to queue

SendEmail
- remove msg from queue
- compose e-mail msg
- send e-mail via e-mail server
3-step architecture design process: Validation

1. **Determine architectural requirements**
   - create a statement or model of requirements that will drive the architecture design

2. **Architecture design**
   - define the structure and responsibilities of the components

3. **Validation**
   - test the architecture against existing and (possible) future requirements
Purpose of validation step:
- increase confidence of design team that architecture is fit-for-purpose

Two approaches to validation:
(1) manual testing using scenarios
(2) ‘Technical’ testing using prototypes
Two approaches to validation: (1) manual testing using scenarios e.g.

<table>
<thead>
<tr>
<th>Quality attrib. for Order proc. system</th>
<th>Stimulus example</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modifiability</td>
<td><em>customer system</em> is updated to an Oracle DB</td>
<td>The Validate component must be re-written to interface to the Oracle-based system</td>
</tr>
<tr>
<td>Availability</td>
<td><em>e-mail server</em> fails</td>
<td>(1) Messages build up in the <em>OrderQ</em> until the e-mail server restarts. (2) Messages are then sent by the <em>SendMail</em> component to remove the backlog. (3) Order processing is not affected.</td>
</tr>
<tr>
<td>Reliability</td>
<td><em>Customer or Order systems are unavailable</em></td>
<td>If either fails, order processing halts &amp; alerts are sent to system administrators so that the problem can be fixed.</td>
</tr>
</tbody>
</table>
Two approaches to validation:

(2) ‘Technical’ testing using prototypes

**Prototype is:**
- a cut-down version of desired application
- created specifically to test some high-risk or poorly understood aspects of the design

**Typically used for two purposes**

(i) **proof-of-concept:** can architecture be built in a way that satisfies the requirements?

(ii) **proof-of-technology:** does the selected technology (middleware, libraries, etc.) for implementing the application behave as expected?
On Friday afternoon, orders must be processed at the rate of 5000 orders in 5 minutes

**NOT possible to validate** if design can satisfy this requirement using scenarios.

=> use prototypes. e.g. of OrderInput prototype:

(1) Write a test program that calls the Customer System validation APIs 5000 times, and test how much time this takes.

(2) Write a test program that calls the Order System store APIs 5000 times, and test how much time this takes.
Outputs of architectural design

(1) **Views**: diagrams that depict the design

(2) **Documents**: explain design, reasons for design decisions & associated risks

**HOMEWORK FOR STUDENT:**

a. Why document the design?

b. Problems associated with documenting the design.
Outputs of architectural design

1. **Architecture Views**: diagrams that depict the design
   - Informal notations are commonly used for the diagrams
   - **UML diagrams**:  
     - class diagrams  
     - component diagrams  
     - sequence diagrams  
     - etc.

are a formal way of presenting views
(1) **Architecture Views:** diagrams that depict the design

- **UML diagrams:**
  - class diagrams, **component diagrams**, sequence diagrams, etc.

  are a formal way of presenting views.

  **e.g. component diagram for order processing example:**

A **component diagram**
shows relationship between components with well defined interfaces.

A **component** typically
**consist of multiple classes.**

**Note:**
OrderInput, Validate, Store are
now in one component.

Another **queue** has been introduced.
(1) **Architecture Views**: diagrams that depict the design

- **UML diagrams**:
  - **class diagrams**, component diagrams, sequence diagrams, etc. are a formal way of presenting views.

  e.g. Class diagram for **OrderProcessing** component:

A **class diagram** shows the classes in the system and their relationships

**Note:**
Only the classes for interacting with other existing systems are shown. (relevant to the architecture design)

=> simplicity
Documenting a Software Architecture (5)

Outputs of architectural design

(2) **Documents**: explain design, reasons for design decisions & associated risks

- Use of **a document template** is advised. e.g.

  **Project Name: pppppppp**
  1. Project context
  2. Architecture requirements
  3. Solutions
     - 3.1 Relevant patterns
     - 3.2 Structural views ..... 
  4. Architecture Analysis
     - 4.1 Scenario Analysis
     - 4.2 Risks

---

Architectural Requirements

Choose Architectural Framework

Allocate Components

Architecture Views

Architecture Document
Standards for Architecture documentation

Some reasons for standards:

- An **organisation can benefit** by defining architecture description (documentation) standards *e.g.*:

  - software architects of an organisation can specify a set of viewpoint definitions which could be re-used from one architecture description to another.

**IEEE 1471 – 2000 : year 2000**

- **Title:** ‘Recommended Practice for Architectural Description of Software-intensive Systems’
- **Purpose:** to establish a common frame of reference for architecture description.

**ISO/IEC 42010 (and IEEE 42010) : year 2006**

- **Title:** ‘Systems and Software Engineering – Architecture Description’
- **Purpose:** Extends the IEEE 1471 by specifying requirements for architecture frameworks
- **architecture framework** = conventions & practices for architecture description established within a specific domain or stakeholder community.
5. Class activities (20 minutes)

Refer to one of the following architecture view UML diagrams and explain what the diagram depicts (one diagram per group)

1. Interactions view: sequence diagram – Fig. 47
2. Deployment view: deployment diagram - Fig. 48
3. Component view: representing interfaces – Fig. 49
4. Component view: using ports – Fig. 50
5. Component view: internal design – Fig. 51
5. Class activities: diagram 1

1. Interactions view: sequence diagram – Fig. 47
5. Class activities: diagram 2

2. Deployment view: deployment diagram - Fig. 48

Fig. 48. UML Deployment diagram for the order processing system
5. Class activities: diagram 3

3. Component view: representing interfaces – Fig. 49
5. Class activities: diagram 4

4. Component view: using ports – Fig. 50
5. Class activities: diagram 5

5. Component view: internal design – Fig. 51

Fig. 51. Internal design of the OrderProcessing component