**CH: Configuration Management:**

*Configuration management activities*

1. We can know the status of change request form from **Decision**
2. **Version management** ⇨ the process of keeping track of different versions
● Show the base line for the given code line:

![Base Line Diagram]

- delta ➞ when the system stores a list of differences between the versions

**Build platform:**

1. In development system ⇒ Developers check out code from the version
2. In build server ⇒ Developers check-in code to the version

**Types of releases:**

1. **major releases** ➞ deliver significant new functionality
2. **minor releases** ➞ repair bugs and fix customer problems

**In Release timing:** IF

- releases are too frequent or require hardware upgrades ⇒ customers may not move to the new release.
- releases are too infrequent ⇒ market share may be lost as customers move to alternative systems.

- Lehman’s fifth law ◄► says if you add lot of new functionality, you’ll introduce bugs. Therefore a system with new functionality may have followed by release + fix that bug

---

**CH: Service oriented:**

✔️ **X** Services are platform and implementation language independent.

**X ✔** Services are language dependent.
(✓ ×) Investment in legacy systems can be preserved.

Service-oriented architecture

Key standards:

1- **SOAP**: (messaging)
2- **WSDL**: service interface and its bindings. (service definition)
3- **WS-BPEL**: for workflow languages. (process)

● different between service & component ⇒ service is independent + Not require Interface.

● from the following write the name of: namespace prefix, some element with its type, one operation

```xml
<types>
  <xs:schema targetNamespace = “http://…/weathns”
    xmlns:weathns = “http://…/weathns” >
    <xs:element name = “PlaceAndDate” type = “pDate” />
    <xs:element name = “MaxMinTemp” type = “mntemp” />
    <xs:element name = “InDataFault” type = “errmess” />
  <interface name = “weatherInfo” >
    <operation name = “getMaxMinTemps” pattern = “wsdlns: in-out”>
      <input messageLabel = “In” element = “weathns: PlaceAndDate” />
      <output messageLabel = “Out” element = “weathns:MaxMinTemp” />
      <outfault messageLabel = “Out” element = “weathns:InDataFault” />
    </operation>
  </interface>
</types>
```
-service engineering process

- **Types of services:**
  1. Utility ➤ general functionality
  2. Business ➤ specific business function
  3. Coordination ➤ support composite processes

- **Classification:**
  1. Task-oriented ⇔ (utility / business / coordination) associated with activity
  2. Entity-oriented ⇔ (utility / business) as object

- (✘ ✔) Notations such as the XML are a more abstract representation than UML.

- **Choose:** Exceptions defines in (Logical interface design / Message design).

- (✔ ✗) the **informal** description of the functionality provided by the service helps users to decide if the service is what they want.

- Graphical workflow languages, such as **BPMN** ➤ more readable, may be used to describe a business process and the services.

- Service construction by composition
Black boxes !!

example of service workflow

CH: reuse:

- **complete**: is a subsystem design made up of a collection of abstract and concrete classes and the interfaces between them ☞ **Frameworks**

- **Frameworks classes**:
  1. System infrastructure.
  3. Enterprise application.

- **software product line** ☞ common architecture & different requirements

- **difference between Application frameworks and product lines**:

<table>
<thead>
<tr>
<th>Application framework</th>
<th>rely on object-oriented</th>
<th>providing technical rather than domain specific support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product lines</td>
<td>need not be object-oriented</td>
<td>embed domain and platform information</td>
</tr>
</tbody>
</table>
**Product instance development:**

- **COTS solution systems** ➪ generic application systems
- **COTS integrated Systems** ➪ use it when there is no single COTS system that meets all of your needs

---

**CH: Testing:**

- **some testing types:**
  1. Unit testing
  2. System testing
  3. User testing

- (✔ ✗) Can reveal the presence of errors NOT their absence.
- **Validation testing** ⇒ the software meets its requirements
- **difference between**
  - Inspections ⇒ static verification and **testing** ⇒ dynamic verification
- inspection & testing

---

*Testing need to execute the program to ensure it meets requirements (dynamic)*
- **inspection advantages**:
  1. No need program execution.
  2. Can implemented to Incomplete versions of a system with no adding cost.
  3. Test portability and maintainability attributes.

- **inspection disadvantages**:
  1. cannot check the non-functional requirement such; performance.
  2. errors can mask (hide) other errors.

- **Stages of testing**: 
  - development such ▶ unit / partition / interface / Automated / Regression / system testing
  - release such ▶ Requirements / testing by scenario / Performance / stress testing
  - user such ▶ alpha / beta (before release) / Acceptance testing (after release)

- **interface type**:
  1. Parameter interfaces.
  2. Shared memory interfaces.

- **interface errors**:
  1. Misuse ⇔ wrong parameter type, wrong parameter order, or wrong number of parameters passed
  2. Misunderstanding ⇔ miss understand the returned value from method
  3. Timing ⇔ problem in communication synchronisation between process

- **complete**: is an approach to program development in which you inter and code development (TDD).

- **TDD activities**:

![TDD diagram]

- Identify new functionality
- Write test
- Run test
- Implement functionality and refactor
• Benefits of TDD:
  1- Code coverage.
  2- Regression testing.
  3- Simplified debugging.
  4- System documentation.

• (✔ ✗) regression testing is more expensive than manual tasting.

CH: Quality Management:

• Quality management and software development:

  - complete: is a record of progress and supports continuity of development as the development team changes (quality documentation)
• **Importance of standards** ⇒ avoids repetition of past mistakes.

<table>
<thead>
<tr>
<th>Product standards</th>
<th>Process standards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design review form</td>
<td>Design review conduct</td>
</tr>
<tr>
<td>Requirements document</td>
<td>Submission of new code for system building</td>
</tr>
<tr>
<td>structure</td>
<td></td>
</tr>
<tr>
<td>Method header format</td>
<td>Version release process</td>
</tr>
<tr>
<td>Java programming style</td>
<td>Project plan approval process</td>
</tr>
<tr>
<td>Project plan format</td>
<td>Change control process</td>
</tr>
<tr>
<td>Change request form</td>
<td>Test recording process</td>
</tr>
</tbody>
</table>

• **Problem with standards:**

1. not be seen as relevant & up-to-date.
2. too much bureaucratic form filling.
3. unsupported by software tools.

**ISO 9001 core processes**
- **Quality reviews** ⇒ A group of people carefully examine part or all of a software system and its associated documentation.

- **reviews types:**
  1- Inspections for defect removal (product);
  2- Reviews for progress assessment (product and process);
  3- Quality reviews (product and standards).

- (✔️ ✗) We can only measure internal attributes.

- **Dynamic metrics** ⇒ assess efficiency and reliability
- **Static metrics** ⇒ assess complexity, maintainability and understandably.

- (✗ ✔️) Static metrics have an direct relationship with quality attributes.

- (✔️ ✗) Reducing the number of faults in a program leads to an increased number of help desk calls.