Quality Control

W3C XML Schema

- DTDs are chiefly directed toward describing how elements are arranged in a document. They say very little about the content in the document, other than whether an element can contain character data. There is no way to constraint the type of data in an element.

- Suppose a census taker submitted a census record with many mistakes in the data entry. For example incorrect date format, some important fields left empty, age impossible etc.

Modularisation

```
<census-record taker="9170">
  <date><month>?</month><day>110</day><year>03</year></date>
  <address>
    <street></street>
    <city>Munchkinland</city>
    <county></county>
    <country>Here, silly</country>
    <postalcode></postalcode>
  </address>
  <person employed="fulltime" pid="?">
    <name>
      <first>Meeble</first>
      <last>Bigbug</last>
    </name>
    <age>1548625</age>
    <gender>yes</gender>
  </person>
</census-record>
```

XML Schema

- It isn't hard to write a programme to check datatypes however it is low level and the idea of a DTD is a formal description of a markup language. So we can conclude that DTDs do not go far enough in describing a markup language.
- To make matters worse, a DTD will reject trivial errors, such as the incorrect order of the date elements. If we were to write a rule to allow the order of day month and year to be in any order and only occur once each then it would require:

```
<!ELEMENT date (
    (year, ((month, day) | (day, month)))
    | (month, ((year, day) | (day, year)))
    | (day, (month, year) | (year, month)))
    )>
```

- Ugly?

XML Schema

- The other limitation of DTDs is the lockdown of namespace. Any element in a xml document must have a corresponding declaration in the DTD. No exceptions.
- To address these problems, a new validation system was invented called schema. Like DTDs schemas contain rules that all must be satisfied for a document to be valid.
- There are several competing schemas, the one that is sanctioned by W3C is XML Schema. Another proposal called RELAX NG, adds capabilities not found in XML schema.

XML Schema

- XML schemas are themselves XML documents (unlike DTDs)
- It is more verbose that DTD but still readable
- From the census examples this is how you would define the county element:

<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema>

<xs:element name="county" type="xs:string"/>

</xs:schema>

The xs:element acts like an <!ELEMENT declaration in a DTD.

xs:string refers to a simple type of element, one that is built in to the schema specification. It's just a string of character data.

XML Schema

- xs:string contains any text
- xs:token contains text but collapses extra spaces
- xs:Qname contains a namespace-qualified name
- xs:decimal contains decimal number
- xs:integer contains integer number
- xs:float contains a 32-bit floating point number
- xs:ID, xs:IDREF same as ID in DTDs
- xs:boolean true or false
- xs:time time HH:MM:SS – timezone
- xs:date CCYY-MM-DD
- xs:dateTime both of above.

XML Schema

- Most elements are not simple however, they contain other elements, attributes, and character data with specialised formats. So schemas contain complex element definitions.

```
<xs:element name="date">
    <xs:complexType>
      <xs:all>
          <xs:element ref="year"/>
          <xs:element ref="month"/>
          <xs:element ref="day"/>
      </xs:all>
    </xs:complexType>
</xs:element>
<xs:element name="year" type="xs:integer"/>
<xs:element name="month" type="xs:integer"/>
<xs:element name="day" type="xs:integer"/>
```

- The date element is a complex type because it has special requirements that you must explicitly define. In this case, the type is a group of three elements (in any order), referred to by name using the ref attribute. These referred elements are defined at the bottom to be of type integer.

XML Schema

- It is possible to refine the date even further. Although the schema will guarantee that each of the subfields year, month, and day are integer values, it will allow some values we don't want. for example, -12345 is a valid integer, but we wouldn't want that to be used for month.
- The way to control the range of a data type is to use facets. A facet is an additional parameter added to a type definition. You can create a new data type for the <month> element:

```
<xs:simpleType name="monthNum">
    <xs:restriction base="xs:integer">
        <xs:minInclusive value="1"/>
        <xs:minInclusive value="12"/>
    </xs:restrition>
</xs:simpleType>
<xs:element name="month" type="monthNum"/>
```

XML Schema

- We created a named type and called it monthNum. Named types are not bound to any particular element, so they are useful if you'll be using the same type over and over.
- the xs:restriction allows us to derive a more specific type than just xs:integer. Inside are two facets, minInclusive and maxInclusive, setting the lower and upper bounds.
- Besides setting ranges, facets can create fixed values, constrain the length of strings and match patters with regular expressions. For example, say you want the postal code to be any string that contains three digits followed by three letters as in the census example.

<postalcode>885JKL</postalcode>

XML Schema

- we want to pattern match it to [0-9]{3} and [A-Z]{3}

```
<xs:simpleType name="pcode">
    <xs:restriction base="xs:token">
        <xs:pattern value="[0-9]{3}[A-Z]{3}"/>
    </xs:restriction>
</xs:simpleType>
<xs:element name="postalcode" type="pcode">
```

- enumeration is another way of defining a set of allowed values.

```
<xs:simpleType name="genderType">
    <xs:restriction base="xs:token">
        <xs:enumeration value="female"/>
        <xs:enumeration value="male"/>
    </xs:restriction>
</xs:simpleType>
```

XML Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- document element -->
  <xs:element name="census-record">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="date"/>
        <xs:element ref="address"/>
        <xs:element ref="person" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute ref="taker"/>
    </xs:complexType>
  </xs:element>

  <!-- Number identifying the census taker (1-9999) -->
  <xs:attribute name="taker">
    <xs:simpleType>
      <xs:restriction base="integer">
        <xs:minInclusive value="1"/>
        <xs:maxInclusive value="9999"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>

  <!-- structure containing date information -->
  <!-- this is a simplification over the previous definition using -->
  <!-- three sub-elements. -->
  <xs:element name="date" type="date"/>

  <!-- structure containing address information -->
  <xs:element name="address">
    <xs:complextype>
      <xs:all>
        <xs:element ref="street"/>
        <xs:element ref="city"/>
        <xs:element ref="country"/>
        <xs:element ref="postalcode"/>
      </xs:all>
    </xs:complextype>
  </xs:element>
```

XML Schema

```
<xs:element name="street" type="string"/>
<xs:element name="city" type="string"/>
<xs:element name="county" type="string"/>

<!-- postalcode element: uses format 123ABC -->
<xs:element name="postalcode"/>
  <xs:simpleType>
    <xs:restriction base="string">
      <xs:pattern value="[0-9]{3}[A-Z]{3}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<!-- structure containing data for one resident of the household -->
<xs:element name="person">
  <xs:complextype>
    <xs:all>
      <xs:element ref="name"/>
      <xs:element ref="age"/>
      <xs:element ref="gender"/>
    </xs:all>
    <xs:attribute ref="employed"/>
    <xs:attribute ref="pid"/>
  </xs:complextype>
</xs:element>

<!-- Employment status: fulltime, parttime, or none -->
<xs:attribute name="employed">
  <xs:simpleType>
    <xs:restriction base="string">
      <xs:enumeration value="fulltime"/>
      <xs:enumeration value="parttime"/>
      <xs:enumeration value="none"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
```

XML Schema

```
<!-- Number identifying the person (1-999999) -->
<xs:attribute name="pid">
  <xs:simpleType>
    <xs:restriction base="integer">
      <xs:minInclusive value="1"/>
      <xs:maxInclusive value="999999"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>

<!-- Age (0-200) -->
<xs:element name="age">
  <xs:complexType>
    <xs:restriction base="integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="200"/>
    </xs:restriction>
  </xs:complexType>
</xs:element>

<!-- Enumerated type: male or female -->
<xs:element name="gender">
  <xs:simpleType>
    <xs:restriction base="string">
      <xs:enumeration value="female"/>
      <xs:enumeration value="male"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<!-- structure containing the name; note the choice element
     that allows an optional junior OR senior element -->
<xs:element name="name">
  <xs:complextype>
    <xs:all>
      <xs:element ref="first"/>
      <xs:element ref="last"/>
    </xs:all>
    <xs:choice minOccurs="0">
      <xs:element ref="junior"/>
      <xs:element ref="senior"/>
    </xs:choice>
  </xs:complextype>
</xs:element>

<xs:element name="junior" type="emptyElem"/>
<xs:element name="senior" type="emptyElem"/>

<!-- Defining a type of element that is empty -->
```

```
  <!-- Defining a type of element that is empty -->
  <xs:complexType name="emptyElem"/>
</xs:schema>
```

XML Schema

- since XML Schema supports a variety of date formats for character data, it makes sense to replace the cumbersome date container and its three child elements with one that takes only text content.
- maxOccurs allows an unlimited number of person elements, without it the schema would allow no more than one such element. (default is 1).
- A choice element is the opposite of all. Instead of requiring all the elements to be present, it will allow only one of the choices to appear. In that case either <junior/> or <senior/>
- minOccurs attribute in the choice is set to zero to make it optional (again default is 1).
- Curriously there is no type for empty elements and so you define one for the <junior> and <senior> elements.