XML in SSC1 versus First year info+web

- Information and the Web is optional in Year 1
- Info+Web is a how-to for web technologies (HTML, XML, Javascript, ...)
- Iain Styles’s lecture notes: “XML: A Toolkit for Structured Information”
- In this module, we have only a few lectures on XML
- But you know more CS now: reg exps, grammar, parse trees
- Will focus on those aspects of XML: structure
- Learning Outcome for SSC1: “structural principles of XML”
Some XML constructs and terminology

- Tags: opening `<a>` and closing `</a>`
- Elements: complete `<a>` ... `</a>` pair
- Attributes inside tags, e.g. `<div id="navigation">`
- Empty tag `<br />` abbreviates `<br></br>`
- W3C produces official standards for XML and its allies
- Why have one standard when you can have many? HTML, XHTML 1.x, HTML5, SGML, ...
- “ML”: markup language: tags
Example: XHTML

- modern HTML standard based on XML
- Separation of structure (XHTML) from layout (CSS)
- Same webpage can be displayed differently (e.g. for printing, partially sighted)
- CSS = Cascading Style Sheets
- CSS is an “unparser” for XHTML
- CSS renders tree as nested boxes
- Nesting of elements ⇒ nesting of boxes
XML = Dyck language + text

Example: module web page structure in XHTML.

```xml
<html>
  <head>
    <title>
      Software Systems Components 1, 2011-2012
    </title>
  </head>
  <body>
    <h1>
      Software Systems Components 1, 2011/2012
    </h1>
  </body>
</html>
```
Example: Module web page with CSS

The XHTML has been parsed into a DOM tree. CSS has traversed the tree and rendered nesting boxes.
Same XHTML. All the info is and tree structure there. Rendered with default old-timey layout; nesting not as obvious.

Software Systems Components 1, 2011/2012

- Hayo Thielecke’s Home Page
- Publications

Teaching 2011-2012

- Secure Programming
- Principles of Programming Languages
- Software Systems Components 1 (SSC1)

PhD and MSc topics

- PhD research topics
- MSc Computer Security topics
- MSc Advanced CS topics
- MSc Computer science topics

Class test

The SSC1 class test will be on Tuesday 18.10. (week 4), in the lecture at 15:00. It counts for 30% of the continuous assessment mark. The test will be based on any material from weeks 1-3.

If you are entitled to extra time during examinations, please contact the TA Phil Smith (pxs697@cs.bham.ac.uk), who will organize a separate venue for the test.

Week 4: XML

Slides on XML
In the browser, the renderer walks over the XML tree. At each node, it applies a CSS style rule. Example: header in navigation bars:

```css
#navigation h4
{
    width : auto;
    padding : 5pt;
    background-color : #2F4F4F;
    margin : 0em;
    color : white;
    margin-bottom : 0pt;
    font-size : 110%;
}
```
XML needs to be parsed
- Match brackets and build tree
- somewhat like a Dyck language parser
- but more complex
- XML parsing makes a distinction between “well-formed” and “valid”
- In grammars, no such distinction (valid is implicit)
Well-formed and valid

Well-formed XML
An XML file is legal or well-formed if it meets certain syntactic conditions.
The most important of these conditions is that all the tags match.
Compare Dyck: [ [ ] [ ] ]

Valid XML
If an XML file is well-formed, it may still not have the structure we expect it to have. For instance, do we want an element A to have a B child? If so, how many?
Compare A → B C
An example for validity

```
<A>
  <B>
  </B>
  <C>
    <D>
      Fnord
    </D>
  </C>
</A>
```

There is additional structure:
A element contains a B and a C
C contains D
D contains plain text
The previous example is valid for the following DTD

```xml
<!ELEMENT A (B, C)>

<!ELEMENT B EMPTY>

<!ELEMENT C (D)>

<!ELEMENT D (#PCDATA)>
```

Note: #PCDATA means text.
Regular expression constructs in DTDs

- DTDs can contain reg exp constructs like + and * for repetitions and | for alternation.
- However, the reg exps have to be deterministic.
- No FIRST/FIRST conflicts are allowed.
- Need to rewrite \((a \, b) \, | \, (a \, c)\) as

\[
a(b \, | \, c)
\]
Recall how we represent *typed* parse trees using the COMPOSITE pattern.

```java
class A { B b; C c; }

class B { }

class C { D d; }

class D { String fnord = "Fnord"; }
```
Lisp has “symbolic” expressions since the 1960s.
Lots of Irritating Silly Parentheses
Idea: forget about syntax, just write down the parse tree
Put in enough brackets to make it unambiguous
Lisp syntax is analogous to XML but more concise
(A ... ) is like <A> ... </A>
Our example becomes:

(A (B) (C (D "Fnord")))
Tree parsers

- Tree is constructed while parsing
- DOM tree: Document Object Model
- Like a parse tree (roughly)
- Compare: ANTLR builds parse tree for grammar
Streaming parsers, StAX

- DOM trees get large
- May consume too much memory (but it depends)
- **Streaming** parser traverses tree without constructing tree as data structure
- Compare: Parse tree traversal ↔ derivation
- StAX: Streaming API for XML
- Stream of events: beginning of element, end of element, plain text
- We could build the tree from that in principle
- Nice example: echo and indent an XML file
StaX example


```xml
  ELEMENT: <item>
  CHARS: Why
  ELEMENT: <em>
  CHARS: WonderWidgets
  END_ELM: </em>
  CHARS: are great
  END_ELM: </item>
```
Alternatives to DTDs

DTD have limitations. More powerful alternatives include XML Schema and Relax NG. The latter has a reasonable syntax based on grammars and regular expressions.
Language for navigation of XML trees.
Like navigating a Unix file tree with paths
/users/hxt/public_html/2011/19343
Navigate to child element
Navigate to all descendants
If you want to go deeper: XPATH is a modal logic for trees
Software security related to this module

- Relevant to our flagship MSc in Computer Security and my Secure Programming module
  

- Software security is not an add-on feature

- not security software

- Security aspects are best learned along with the main material

- We should cover some in undergrad teaching, not just the MSc
Streams and security

- Serializing can be dangerous.
- Loading an object of unknown class $\Rightarrow$ Dynamic class loading $\Rightarrow$ Malicious mobile code
- Calendar Bug in Java:
  http://slightlyrandombrokenthoughts.blogspot.com/2008/12/calendar-bug.html
Regular expression matching and security

- Inefficient backtracking for reg exp matching takes forever on some inputs.
  Attackers can force a Denial of Service (DoS) attack.
- Regular Expression Denial of Service Attacks and Defenses
XML and security

- Attackers can force a Denial of Service (DoS) attack.
- XML Entity expansion
- XML parsing: Loads of opening tags overflow the parsing stack.
- XML injection; XPATH injection
- compare SQL injection (SSC2)
The stream hierarchy cooks raw input, e.g., 12345 → integer
Regular expression matching can be done on streams
Regular expressions ≠ Perl “regex” (which deserve sneer quotes) with exponential runtime
Parsers are more powerful than regular expressions
Regular expression constructs can be used in grammars
Regular expressions are also part of other languages, e.g., DTDs for XML
XML needs to be parsed; it is a language of brackets like Dyck
XML parsers may build a tree or not, just like general-purpose parsers