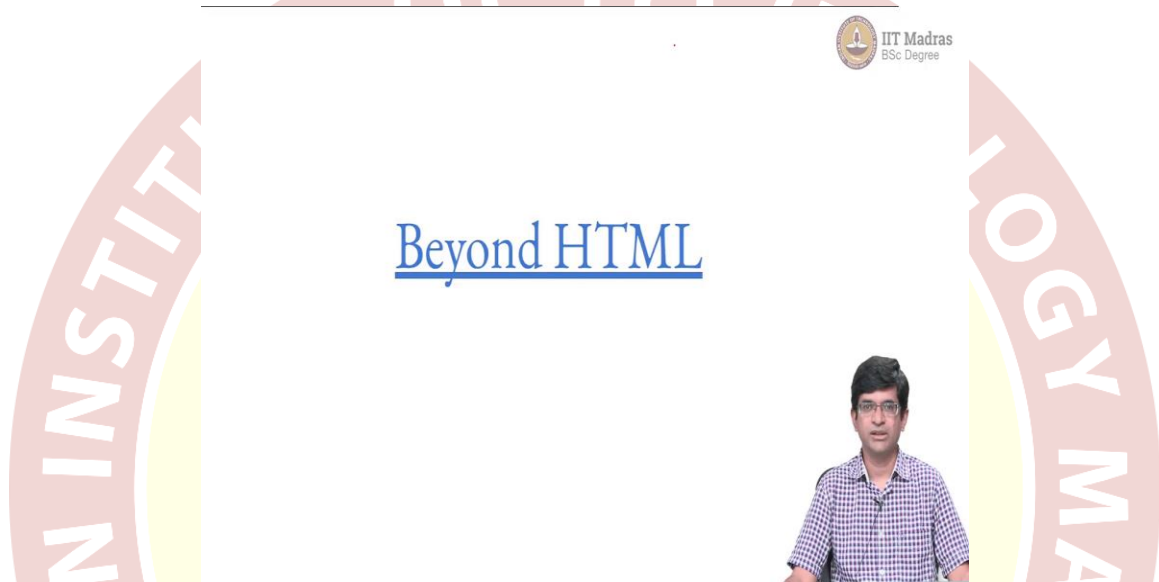


**IIT Madras**  
ONLINE DEGREE

**Modern Application Development**  
**Professor Nitin Chandrachoodan**  
**Department of Electrical Engineering**  
**Indian Institute of Technology, Madras**  
**Beyond HTML**

Hello, everyone, and welcome to this course on Modern Application Development.

(Refer Slide Time: 00:18)



Hello, everyone. We are now getting to one of the more advanced topics or towards the end of this course on Modern Application Development, part 1. And in some sense the topic that we are covering in these lectures, which I have titled Beyond HTML, is essentially something which as an app developer you need to know, but on the other hand is also something which does not directly impact the development of your application.

It is more important from the point of view of understanding what comes next. So far you have been working with the basics of html, css, but nowadays applications are much more dynamic. And there are a lot of ways in which you can create more interactive and more user friendly applications. And in some sense as you move forward in the area of application development, that is the direction in which you are going to go.

So, it pays to sort of know a little bit about what comes next, what comes beyond html, how do you extend html, what are the ways in which you can improve the functionality in your application, and that is basically what we are going to look at here.

Now I also need to add that this is probably one of the more controversial set of topics that we have. Controversial, not in the sense that there is anything wrong with it, but that many of the topics over here are very subjective, and people have strong opinions saying that one framework or one language or one way of doing things is the correct way of doing it.

On the other hand, the approach that we have tried to take in this course is that we cannot really teach you all possible frameworks, and there is no such thing as the best possible framework for the simple reason that different people have different styles of writing code.

So, the best that we can try to do is to try and teach you the fundamental concepts and hopefully whenever you see anything new, any new framework or anything else, you should be able to view it in that context of the different concepts that are there, and figure out whether this works for you or not.

So, with that in mind please keep that clearly in the picture. Some of the concepts over here of course, are fundamental. You need to understand them in order to see how to move forward. But some things which come later in this lecture, such as, for example, the different frameworks that are available are very subjective and different people tend to have strong opinions on which one is better or which one is not suitable for a given purpose.

I am not going to get into the relative merits or demerits of the different frameworks. All I am going to do is present a few choices, and as far as possible focus on the fundamental techniques. And then it is up to you to sort of see what works best for you.

(Refer Slide Time: 02:55)

## HTML Evolution



So, we are going to start with a word on the evolution of the HTML markup language. Once again, I would like to emphasize that HTML is not really a programming language. It does not have control flow constructs, it does not have if then statements, so to say, which allow it to sort of execute loops which are required for arbitrary functionality.

It is meant for a specific purpose, which is markup, and it does that job pretty well. Having said that there were a number of reasons why people wanted to change it or improve upon it as time passed. So let us look briefly at how it evolved to the state where it is now, and to try and understand where we are now so that we know what is likely to happen in the near future.

(Refer Slide Time: 03:40)



### Markup languages

- Origins from late 60s
- Mostly used for typesetting and document management systems
- Problems?
  - Lack of standardization
  - Target audience: coders, publishers, academics?
  - Target output: print, other forms of media
  - Machine readability



So, before talking about HTML itself, let us look at the general idea of markup languages. Now, the idea of markup, which is to say that you add some specific lines or text somewhere within the text of a document, and those specific tags or keywords are interpreted in a certain way and convey meaning either to the reader or to some kind of a, another application, which could then look at those tags, understand that you wanted to emphasize or itemize certain points over here, some information which is over and above what is clearly present from just the text itself.

So that was the idea behind markup language. And it originated pretty much in the 60s. So 50 years old, more than 50 years old at this point. Mostly it was used for typesetting and for some document management systems. One of the bigger developments in this area was a document management system at IBM, and one of the people who developed it is generally considered one of the major forces behind the development of markup languages, in general.

The problem that happened, of course, was just like any new technology there was no, nothing standardized about it. Each person who sort of investigated this topic came up with some ideas of their own.

And depending on your target audience were you writing, were you trying to create something for coders or programmers, or was it for publishers who are trying to bring out

magazines, newspapers, something of that sort, or is it, was it for academics, and who might be interested in, for example, mathematical equations, figures, type setting with different fonts, putting things in two columns, suitable for journal publications, and so on.

Depending on your target audience, there were different needs that were being addressed by the markup languages. On top of that there were different output media also. One was, of course, the most popular one was probably just print. That is to say, you needed to have A4 or letter size paper onto which you needed to print out the information.

On the other hand, you might be also targeting newspapers which have a different structure and different page size, or you might be looking at some kind of brochures or you might be looking at something completely electronic. Electronic media in the 60s was not as prevalent as it is now, but even then it existed.

So, depending on the target output you might have different requirements for your markup. And one of the important criteria that people were thinking about in terms of markup languages was, there is this concept of machine readability, meaning that it is, the markup is there not just to tell a human being what you are trying to do but also to indicate to the machine what the different parts of the text correspond to. So that the machine, the computer could automatically, for example, from a document, extract a table of contents. Things of that sort could be done if your markup was good.

(Refer Slide Time: 06:55)

## SGML

### Standard Generalized Markup Language

- Meant to be a base from which any markup language could be designed
- Basic postulates:
  - Declarative: Specify structure and attributes, not how to process
  - Rigorous: strict definition of structure, like databases
- DTD - Document Type Definition
  - Used to specify different families within this umbrella
  - Each could have its own tags, interpretation
- SGML *Applications*





So, the first sort of standardized approach to this was something called SGML. Well, maybe not the first but definitely one of the biggest, sort of, pieces of work in this direction. SGML stands for Standard Generalized Markup Language.

It was meant to be a basis from which any markup language could be developed or designed. In other words, it was sort of a meta language. What I mean by that is you would not normally just say that I have written a document in SGML. I mean, like, for example, you would say that I have written a document in html. That makes sense.

But it does not really directly make sense to say that I have written a document in SGML because SGML does not specify exactly which tags are allowed to be used. In particular, it allows you to create tags of your own. And what kind of tags are being used is ultimately defined by something called a document type definition. So this DTD was a very important part of SGML. I say was, but SGML still exists and is used in various places. So it still is an important part of how you think about such markup languages.

And ultimately the thing was that people were able to define a variety of SGML applications. So what is an SGML application? It basically says that this document, which follows this application, would have a certain set of tags and needs to be processed in a certain way.

So, this is how you understand what this particular document corresponds to, which means that a compiler or some other program could then process that document and create output which would either be suitable for another program to take, or for a human being to read.

Now, it had a couple of basic postulates on which it was designed. One of them was that it was something called a declarative language. So declarative is something which you might come across more and more as you sort of look at different styles of programming. And declarative programming has an interesting idea. It basically says that you want to tell what you expect the output to be, not how you want that output to be created.

So, the difference is, essentially, saying that if you just go and tell somebody, I would like something like a magazine cover with a large headline and a picture on the front. That would be a declarative way of saying it, whereas an imperative way would be to say

take an A4 sheet, create text with this font size, place it at 1 inch from the left hand margin, and 1 inch from the top margin.

In other words, the imperative way would state precisely what are the steps to be taken in order to achieve the final result, whereas declarative would sort of give you information about what is desired and how exactly that is achieved is then up to another program or another application to figure out.

Declarative is, in some sense more friendly to humans because if I can get by with a declarative way of programming or of specifying something, it means that I can think at a higher level of abstraction. And the details of how it gets implemented are left to another program.

On the other hand, SGML is also rigorous, meaning that it was meant to have a very definite document structure. There are ways in which tags can be opened, ways in which tags can be closed, ways in which tags can be nested, what kind of tags are permitted in different places, all of that was clearly defined, which meant that it could actually hold fairly complicated data structures, or even it could be thought of, at some level, even like a database that could be used to store information.

So those are some of the postulates behind SGML. As you can imagine this, kind of generality that people are looking for, while it is a great idea, has a flip side. And usually that flip side is that it makes things difficult to implement, either to implement or to understand, even.

So, for a lot of people SGML ended up being too complex to really appreciate, and people started looking for shortcuts. Is there a simpler way that I can do this? Do I really need to go and make a complete document type definition? Do I need to have all of these tags? What happens if I make a slight mistake here or there? It should be a bit more forgiving. So that was the sort of idea behind why people could not really stick to SGML.



(Refer Slide Time: 11:30)



## HTML

- Originally intended to be an *application* of SGML
- Very lenient with parsing - meant to be forgiving of errors
  - Not valid SGML
- HTML 2.0 - attempt to become SGML compliant
- Legacy support
  - Not truly SGML compliant
- HTML4 official definition - true SGML application
  - Limited usage
- HTML5 - **not** an SGML application - defines its own parsing rules



Html, when Tim Berners-Lee came up with the idea of the web, he essentially intended html, the hypertext markup language to be one specific application of SGML, meaning that it would have a set of tags which was defined as a particular subset or an application of SGML.

But on the other hand, he realized that in order for it to be usable, you cannot really be very strict with those tags. If you keep telling people that they have made a mistake in what they have done over here, they are not going to write HTML documents. So he took a very pragmatic approach and made it very lenient with parsing, which meant that it is forgiving of errors.

So, in other words, if you have not closed a tag properly, or you use a tag where it was not meant to be used, it would really not complain at all. The problem is it meant that it was not valid SGML at that point, which meant that many of the applications or the other programs processing tools that were available for something like this could not really be applied for HTML because they would try to parse an HTML document, find that it did not really follow the specification, give an error.

But as far as the HTML browsers were concerned, that was perfectly valid and you can go along with it. Over time that became only more and more complicated. So when people saw that this was happening, they tried to bring it back into, on track. So HTML

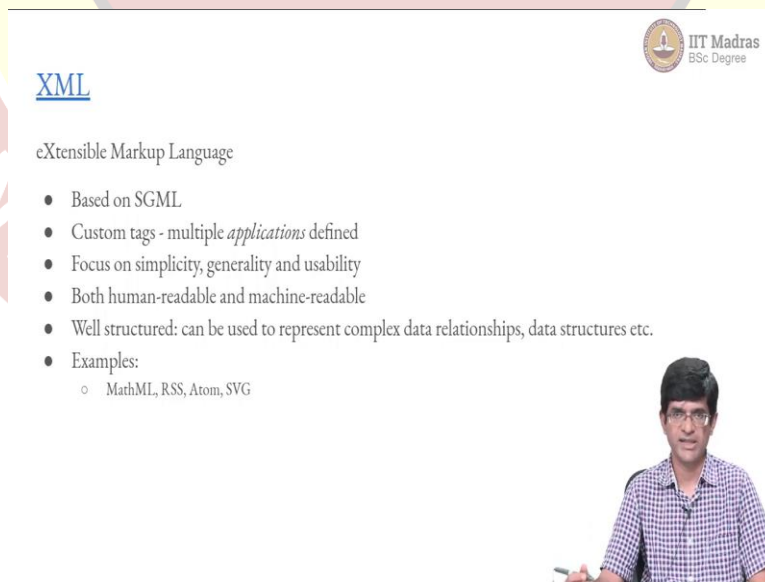
2.0 was one attempt at becoming SGML compliant, but they still needed to support legacy.

Even though this was only like two, three years into the development of the web, it still meant that there were enough web pages that would break if you tried to enforce SGML compliance. So that never really happened.

So finally, HTML4 was the first place at which people could come up with something which was reasonably close to a true SGML application. But on the other hand, it still ended up having to have backwards compatibility. And to say that you cannot just break all the web pages that already exist. You can sort of try and guide people towards writing new pages in a good way, but you cannot break something that already exists. And with that in mind it was always sort of a very tricky situation to use something like HTML 4.

So finally, HTML 5, which we will get to later, just broke away from it completely. It said it is no longer an HTML application. It defines its own parsing rules, which meant that it could automatically, just specify where it can be lenient, where it needs to be strict, what are things that need to be specified properly, and so on. But we are getting ahead of ourselves. I will get back to HTML5 in a moment.

(Refer Slide Time: 14:05)



The slide features a large, faint watermark of the IIT Madras logo in the background. In the top right corner, there is a small circular logo for 'IIT Madras BSc Degree'. The main content of the slide is titled 'XML' in blue underlined text, followed by 'eXtensible Markup Language'. Below this, there is a bulleted list of features: 'Based on SGML', 'Custom tags - multiple applications defined', 'Focus on simplicity, generality and usability', 'Both human-readable and machine-readable', 'Well structured: can be used to represent complex data relationships, data structures etc.', and 'Examples:'. The 'Examples' bullet point has a sub-bullet 'MathML, RSS, Atom, SVG'. In the bottom right corner, there is a small video inset showing a man with glasses and a checkered shirt speaking.

XML

eXtensible Markup Language

- Based on SGML
- Custom tags - multiple *applications* defined
- Focus on simplicity, generality and usability
- Both human-readable and machine-readable
- Well structured: can be used to represent complex data relationships, data structures etc.
- Examples:
  - MathML, RSS, Atom, SVG

So, somewhere along this line, people started to realize that SGML is getting too complex, even it is difficult for human beings to understand it beyond a point. So then

they came up with this thing called XML, which is an extensible markup language once again, based on SGML.

It allowed you to define custom tags, and multiple applications, once again, similar to what SGML does. It had a strong focus on simplicity. At the same time, also on generality, which means that it should be usable in many different contexts, and usability.

So that it got to the point where essentially it started taking out the more difficult parts of SGML and making it a bit more of a pragmatic approach, a practical approach to implementing things, something where you say, okay this might be the right way of doing it but it is just difficult to use. So no, people are not going to do it. Let us make a bit of a compromise over here. Do not completely compromise and say people can do anything but at least simplify it a little bit so it becomes easier to use.

XML is nice. It is both human readable as well as machine readable, and is in fact used for conveying information between machines as well as, it is readable by humans at least. It is not, not, not normally used for conveying information between humans, but is at least readable and writable if needed. It is well structured and can be used to represent complex data relationships and so on. There are several examples of XML currently in use.

(Refer Slide Time: 15:36)

### XML Example - RSS feeds

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
<channel>
  <title>RSS Title</title>
  <description>This is an example of an RSS feed</description>
  <link>http://www.example.com/main.htm</link>
  <copyright>2020 Example.com All rights reserved</copyright>
  <lastBuildDate>Mon, 06 Sep 2010 00:01:00 +0000</lastBuildDate>
  <pubDate>Sun, 06 Sep 2009 16:20:00 +0000</pubDate>
  <ttl>1800</ttl>

  <item>
    <title>Example entry</title>
    <description>Here is some text containing an interesting description</description>
    <link>http://www.example.com/blog/post/K/link</link>
    <guid isPermaLink="false">7bd204c6-1655-4c27-ae4e-53f933c5395K</guid>
    <pubDate>Sun, 06 Sep 2009 16:20:00 +0000</pubDate>
  </item>
</channel>
</rss>
```



One example is you might have come across this notion of RSS feeds. RSS is basically really simple syndication or something that, essentially the idea behind syndication is that anybody who, for example, is publishing let us say a blog, or something where they post items regularly would like to have, make it easy for other people to know that they have updated their website.

And as a user I might want to know when, for example, somebody whose writing style I like, has updated their website. So I can subscribe to something called an RSS feed. And an RSS feed essentially gives me information about what are the latest sort of updates on a given page, and maybe a small blurb about them. And this is an example of what an RSS feed could look like.

So, XML was used as a basis for defining RSS feeds. You can see that the first line over here basically tells you that it is a format of, it is a form of XML. And then the rest of it sort of defines a bunch of tags. So you can see that there is a tag called RSS.

So, if you think about it is it really possible that XML, the people who are defining XML knew that there was something called RSS that is going to come and that inside RSS there would be things called channels which would need to have, for example, a publication date associated with them? No.

So, XML did not care about that. That is where it says that it allows you to design your own tags. And what XML allowed you to do is it has a very neat structure, there is an RSS and a close tag RSS. Within that, there would be a channel, which has a title and a description, and also some kind of a link to the actual webpage of the channel, a few other pieces of information, and then probably a bunch of items in that channel.

So, this would be one example entry. Some text containing an interesting description, some blog post blah, blah, blah, and I would probably have another item after that, and another item. So this channel would contain sort of a set of different items.

Now those of you who have been looking at this carefully might think, hey this can be done with something like JSON. And you are exactly right. JSON was a much later development which precisely sort of takes the place of XML in some ways.

It can mostly replace the XML in most, in many contexts, but on the other hand, XML is very well structured. It actually has even better parsing properties and so on, than what JSON has. It was designed in a certain way, JSON solves the same problem in a different way.

Which one is better? This is once again one of those slightly controversial topics which, at the end of the day both have their place, both serve similar purposes. So which one to use, finally comes down to the simplicity of, and the tools that you have available to you.

(Refer Slide Time: 18:37)

XML Example: SVG

```
<?xml version="1.1"
width="300" height="200"
xmlns="http://www.w3.org/2000/svg">
<rect width="100%" height="100%" fill="red" />
<circle cx="150" cy="100" r="80" fill="green" />
<text x="150" y="125" font-size="60"
text-anchor="middle" fill="white">SVG</text>
</svg>
```

IIT Madras  
BSc Degree

So, another example of XML, you can see that now the set of tags is different. It says SVG. And within SVG it has something called a rect, it has a circle, it has text. What is going on over here? This is, SVG stands for Scalable Vector Graphics. And what it allows you to do is to basically define graphic images. So it is SVG, it has a version number, width, xmlns.

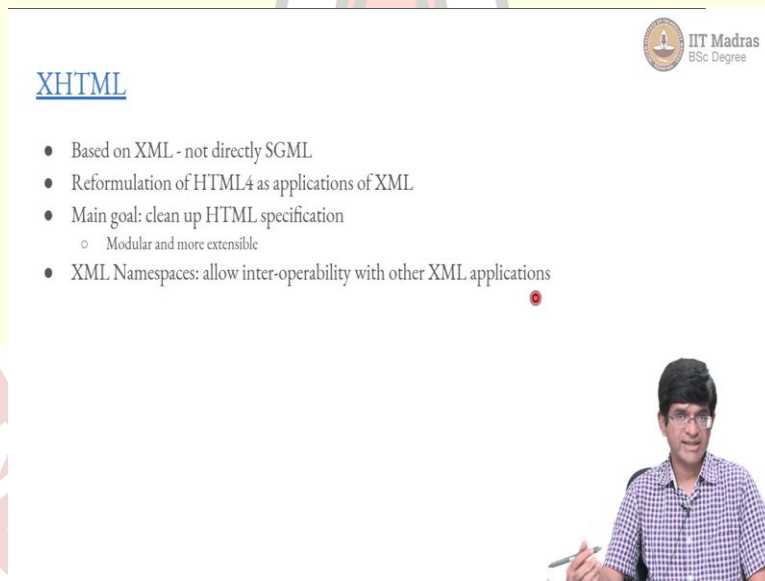
So, this xmlns is the XML namespace. And this XML namespace is sort of the document type definition which tells you what are the types of tags that are permitted in this document. It allows rectangles to be drawn, so you have a red rectangle, you can have a green circle, filled with green, and in the middle of that you can have text which basically says SVG.

So, the thing about this SVG language is that it allows you to specify certain drawing primitives, based on which you can construct larger images. The nice thing is they decided to choose an XML based format for this.

Now you might be wondering where else is XML used? The .docx format, which is used in Microsoft Word, is now an XML based format. The .odt which is used by open document, or the LibreOffice document format is based on XML.

So, all of those, in fact, if you take a .docx file, and you unzip it, I mean, you do not think of it as a zip file but it is actually just a zip file containing multiple other files. If you unzip it, inside it you will find that there are actually multiple XML files. And you can look at each of them directly. So the power of that markup has been taken in many different directions and standardized in different ways.

(Refer Slide Time: 20:30)



XHTML

- Based on XML - not directly SGML
- Reformulation of HTML4 as applications of XML
- Main goal: clean up HTML specification
  - Modular and more extensible
- XML Namespaces: allow inter-operability with other XML applications

Now XML by itself is still a bit complicated, and overly broad for the purposes of the web. So people said, okay, as far as the web is concerned we are looking specifically at hypertext. Can we get something which is based on XML, but is targeted specifically at the web? And that is how XHTML came about.

It is based on XML, it is sort of a reformulation of HTML 4, as an application of XML, instead of SGML. It had the goal of cleaning up the HTML specification and becoming more modular and so on, and is actually a very neat and clean way of doing things. It uses

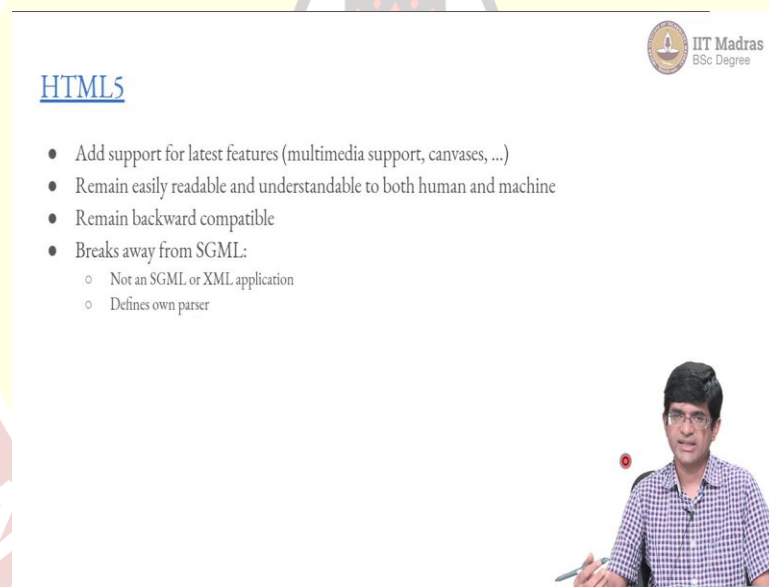


XML namespaces, which means that it allows some amount of interoperability with other XML applications.

So, it looks as though this sort of solved most of the problems but ultimately there was still this issue that you are still trying to adjust with other things. You have this XML which has been defined for a specific machine to machine communication purpose. On top of that you are building HTML, which is meant for, sort of, human communication. And at the end of the day you do end up with some restrictions.

It makes it a bit more difficult when you want to add extra features. XML has its own restrictions on how you can do certain things and so on. XHTML was very close to sort of the ideal scenario that we want.

(Refer Slide Time: 22:00)



HTML5

- Add support for latest features (multimedia support, canvases, ...)
- Remain easily readable and understandable to both human and machine
- Remain backward compatible
- Breaks away from SGML:
  - Not an SGML or XML application
  - Defines own parser

IIT Madras  
BSc Degree

But around that time there was already a parallel development that was going on, which said that we want to have another format called HTML 5. Why did this come about? Because people felt that at the end of the day part of the problem with all these SGML and XML based approaches is that you are trying to be very strict with regard to how the parsing happens because you need to follow all that legacy of the XML or HTML.

And at the same time adding certain support for common features on the web like multimedia, video tags, audio tags, canvases for drawing and so on, would have been

possible next HTML but would have required somewhat complicated tags and extra constructions.

So, people said okay, let us just simplify this, add support for these latest features, remain easily understandable, because that was one of the beauties of HTML, it was so easy that you could actually write it almost like text, whereas XML, all said and done, once you start defining the tags and the type definitions, and so on, starts to get a little complicated.

HTML 5 also wanted to remain backwards compatible, but finally what they said is they want to break away from SGML, and say it is not going to be an SGML or an XML application directly. Most importantly it defines its own parser. So unlike SGML and XML, where the parsing, that is to say, how you read the document and how you interpret the document is very strictly defined, HTML from the beginning was never like that.

So HTML5 finally said let us break that. Let us not, sort of, try and enforce this thing where you have to have that strictness. But if you are going to relax it, let us be clear about where it is being relaxed. And they came up with their own sort of definition of how to parse a document, what things can be accepted, what things should not be accepted and so on.

Now, the interesting thing is HTML5 has this notion that it is the last version of HTML. Why is that? Because at some point there was a lot of controversy, once again, in terms of the development of HTML5. There were two groups who were working on it. One was the W3C, the world wide web consortium, and the other was WHATWG, so the Web Hypertext Application Technology Working Group.

The interesting thing is this WHATWG was sort of, I mean, I am not sure how it is pronounced. I am saying WHATWG, maybe there is a different way of saying it, but this is a group that consisted of people from Mozilla, Google Chrome, Apple Safari, Microsoft Internet Explorer, in other words, all the people making big browsers.

So even though the W3C had a lot of other members who were interested in the applications of hypertext and so on, the browser vendors finally got together and said look we need to be able to add certain kinds of features and the way you guys are going

about it is not fast enough or convenient enough for our liking. So we say that this is the way that need needs to be done.

So, as you can imagine at some point something had to give, and at as of now WHATWG is the one that maintains what is called the HTML Living Standard. So the HTML Living Standard is a web-based document, which you can actually go and look at and read through.

The point is since it is now called a living standard what they say is it can be updated at any point, and whenever you say that something is HTML compliant, rather than saying it is HTML5 or HTML5.1, some version number, you just say that it is as of this date, it is compliant with the tags that are specified over there.

So, that is the sense in which HTML5 is the last version of HTML. It is not that everything about HTML is perfect, but at some level they have basically decided that let us, we have got most of the things right, let us standardize on this.

(Refer Slide Time: 26:02)

### Extension?

- How to add new features? New tags?
- "Software defined"
  - Allow new tags to be added through JavaScript
  - Custom Elements - API supported by browsers
- Very powerful mechanism: arbitrary functionality possible
  - No new tags need to be brought into standard
- Potential problems:
  - Anyone can define a tag!?
  - Semantics (meaning) of tags may not be well thought out

Requirement? Javascript



And then comes the question how do you extend it? That is to say, how do you, I mean what if you come across some new requirement which needs a new tag? How do you add new features? That is where they brought in this notion which, at least in other domains, is usually called software defined, where basically what you say is that I should be able to define new features to my language or my application through some software constructs.

In this case, it allows new tags to be added through the use of JavaScript. What is JavaScript? We will, once again go through that in a little bit more detail, but the important point is there is a mechanism that is defined for extending tags. And you can in fact define so called custom elements through an API.

What is an API? An Application Programming Interface, which basically means that the browser needs to support that API. What that in turn means is as long as the browser understands what these JavaScript function calls are telling it to do, the browser will be able to then execute the JavaScript, then look at a tag and say, okay now, this is what this tag is trying to display on the screen, and this is how I should do it.

It is a very powerful mechanism. Arbitrary functionality is possible because you have now linked a complete programming language into the problem of defining new tags, which means that you do not need to add any new tags into the standard, and whoever wants to can define tags of their own.

This of course, also leads to problems. And the basic fundamental problem you might think about is now this means that anyone can define a tag. If I want to, I can define a new tag of my own. Now, is that a problem? If it is only on my own website, it is not. But it means that the semantics of tags may not be well thought out. Why is this important?

Now, imagine that you are trying to write a web search engine, the next competitor to Google. What does Google do? It reads through webpages and it extracts semantic information from the page. So the page might have, for example, keywords, or it might have a title, it might have H1 headings, H2 headings, and some other things, quotes.

Based on all of that, Google is able to extract certain information about it, what does the table of contents look like, what are the keywords, what is the main, sort of, gist of this page. That can be extracted from the structure of the page.

If I define my own tags, there is no way that a search engine really knows what my H1 tag means. It does not know that it is a heading anymore. So that can lead to a lot of problems. Anyway, so the bottom line is that in order for this particular way of extending HTML to work, JavaScript becomes a requirement.

So, what is JavaScript? Why do we need it? What do we understand from it? That is something that we need to, at least have, a very brief refresher on, and we will talk about that in a little bit of detail.

