System & Network Engineering

Shiny LaTeX for SNE

Azad Kamali
Cedric Van Bockhaven
Leendert van Duijn
Esan Wit

September 23, 2013

Contents

1 XML and XHTML

1.1 W3C Procedure for Recommendation

The W3C procedure for recommendation is defined as follows: a working group creates a *Working Draft* (which indicates the work in progress). This draft is refined to a *Candidate Recommendation*. The *Candidate Recommendation* is further refined to become a *Proposed Recommendation*. A committee decides on whether or not a *Proposed Recommendation* becomes a *W3C Recommendation*. In between the stages there are rounds for feedback in which people from the community/industry are invited to criticize and help improve the document. This process is represented by image 1 below.

1.2 IBM Technical Series

For this assignment we had to watch three videos that can be found on YouTube:

- An Introduction to XML: The Basics [1]
- An Introduction to XML: XML and Web 2.0 [2]
- An Introduction to XML: Managing XML Data [3]

We extracted the following insights from the videos about XML, of which the advantages can be found in table 1.

- XML has a widespread usage everywhere in the industry and is a fully defined, open standard.
- It can be used as a meta language (a language describing the language itself). XML enables easy sharing data between applications, independent of platform or software.
- RSS and ATOM allow to syndicate content to users, ATOM features reusable elements while RSS doesn't.
- AJAX allows the creation of dynamic web applications.
- RDF is the key-language in semantic web development and is XML-based. It describes resources.
- XML allows semi-structured or unstructured data, handles nested and complex data, and it's extensible
- XQuery and XPath allow to search XML data for specific fields, structures or information

- XSLT specifies how to transform XML documents into other XML documents (e.g. applying stylesheet to render XML in browser)

Short overview of advantages and disadvantages

Advantages	**Disadvantages**
Widespread usage	Larger size XML records
Platform independent	Implementation cost
Allows for (un)structured data	Higher complexity for retrieval and storage
Self-definable language	

Table 1: The advantages and disadvantages of XML

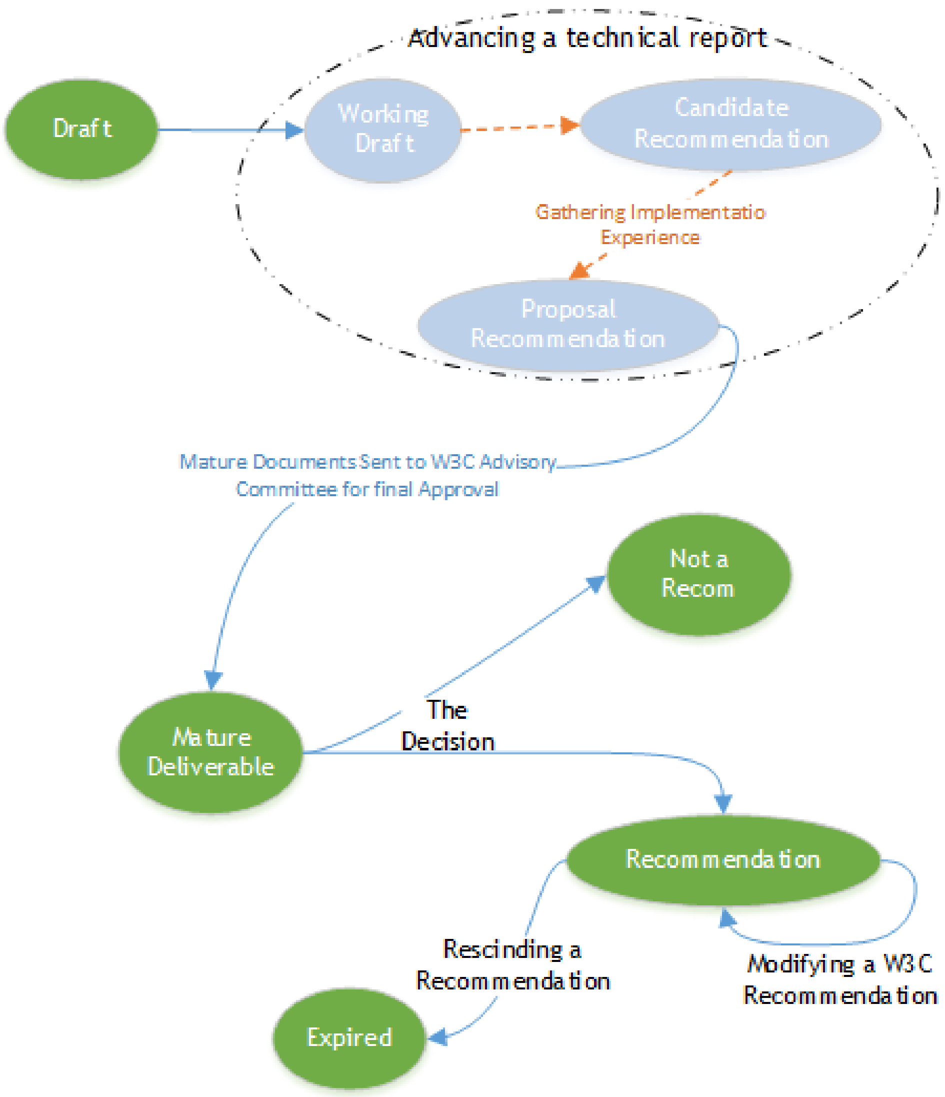
Advancing a technical report
Draft
Working Draft
Candidate Recommendation
Gathering Implementatio Experience
Proposal Recommendation
Mature Documents Sent to W3C Advisory Committee for final Approval
Not a Recom
Mature Deliverable
The Decision
Recommendation
Rescinding a Recommendation
Modifying a W3C Recommendation
Expired

Figure 1: An overview of the W3C Recommendation procedure. Image by Azad Kamali.

2 Further Reading, XML and XSLT concepts

2.1 Transforming XML with XSLT

I found the following points in the Transforming XML with XSLT interesting:

- When processing a document of XML, only a single matched template is used to process the current node.
- If a stylesheet uses only the root template, then it can optionally use the simple form stylesheet syntax that allows `<xsl:stylesheet>` and `<xsl:template match="/">` to be left out.
- It is possible to use multiple templates rather than s single root template.
- We can reuse the previously built templates by using `<xsl:import mode>`.
- It is possible to use a sort of regular expression for matching rules.
- When you create a template, in addition to the match pattern, it can also have a mode attribute that assigns a name to the special mode in which you want to invoke the template.
- When applying multiple rules to one object, the most specific one wins.
- The basic scheme for determining which templates are more specific than others is as follows, ordered by ascending specificity:
  - The generic pattern: `*`
  - Element name: `SOMETHING` or `xyz:SOMETHING`
  - Element path: `SOMETHING/SOMETHINGELSE`
  - Element with matching predicate: `SOMETHING[predicate]`
- All XSLT transformations process the source node tree to produce a tree of result nodes. If multiple transformations are being applied in sequence by your application, the result tree of one transformation becomes the source tree of the next transformation in sequence. When no more transformations need to be done, the final tree of result nodes needs to be written out as a stream of characters again. This process is called serializing the result tree.
- XSLT 1.0 supports three different output methods:
  - `<xsl:output method="xml"/>`
  - `<xsl:output method="html"/>`
  - `<xsl:output method="text"/>`

2.2 Group Discussion Result

Key point to the XSLT parsing is the XPath pattern matching system used to find and match source blocks with the templates. XSLT is modular and can be extended; when using multiple XSLT templates the template with a `"match"` that is more specific then others is used to parse the source node. Templates with a higher specificity take precedence over less specific templates.
E.g. match="ROW" has a lower specificity than "match=ROW[$SAL > 2000$]" as such if "match=ROW[$SAL > 2000$]" is applicable the template which matches merely "ROW" is ignored;
This is different to the process of CSS; in CSS if multiple patterns match an element all rules that are defined for those patterns are applied to the element. If multiple of the applied CSS rules define the same properties the more specific rule overwrites the properties set by less specific rules. But non overridden properties are inherited from which ever rule defined them. In XSLT only the most specific template is used and less specific templates are not inherited;
XPath has support for certain mathematical operations. E.g. doing something special for each 5th row or special handling in case some input value is odd or even;
XSLT transformations can be chained. That is to say if multiple transformations are being applied in sequence of each other then the node tree which is the output of one transformation is used as the input of another transformation. This process is called serializing;
While making a stylesheet in order to make it easier to extend or edit your templates, one can make use of multiple templates rather than a single rooted template stylesheet.

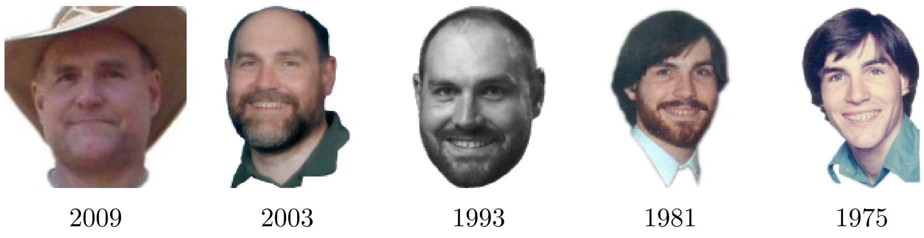

Figure 2: Peter Miller over time

3 Auto-Tools

3.1 Introduction to the Autotools

Introduction to the Autotools is a series of tutorial videos made by David A. Wheeler[4, 5, 6]. Autotools is a collection of various tools designed to make the use of compiling source, and Make in particular, easier [4]. Autotools can help creating platform independent sources [4]. AutoMake, part of Autotools, assists in solving dependencies of source files. It also supports the use of nested structures without the user having to create complicated makefiles [5]. Autotools can be used to help cross-compilation on platforms [5]. Autotools are bundled with methods to resolve and locate dependencies on a system. Autotools supports the recursive make paradigm (although this should not be used, see 3.2) [6]. It's advisable to prefer a large Makefile.am as opposed to a recursive build structure [7, 6].

3.2 Recursive Make Considered Harmful

Peter Miller wrote an article about the problems he noticed in the usage of Make[7]. Most notably problems that originate from the Recursive Make paradigm and incorrect usage/understanding of Make. Image 2[1] shows him over the years.
The author describes several problems with recursive make and attributes these to make not being allowed to do it's work. Make resolves dependencies to determine what requires rebuilding and what doesn't. By the principles of recursive make this becomes impossible. Instead the author propagates the use of full project makefiles of a modular format. A user must then rebuild the entire project. This allows make to properly analyze the DAG for dependencies and determining which parts need to be updates.
The indexing cost of this is negligible when compared to the cost of overbuilding or rebuilding in order to prevent bugs and race conditions. This cost becomes even more negligible when the user takes into account the time saved with debugging. The recursive make paradigm is prone to several bugs and race-conditions. By removing recursive make from the equation time

[1]Source: `http://miller.emu.id.au/pmiller/`

can be saved during development. E.g. time spend debugging the system as opposed to debugging the code.
The full project make also allows make to make full use of the option of parallel building. Because the entire DAG is available make can work on both ends simultaneously, with recursive make this is impossible because make is not aware of the greater scope of the dependencies.

3.2.1 Present Day

A lot of the reasons and arguments in favor of recursive makes are no longer applicable on current computers. Memory has become cheap and processing power too. With current design philosophies going towards greater parallelism it seems that the argument to allow make a full DAG are still applicable. The assumed un-manageability of big makefiles is stopped by the appearance of build tools which automate the process.

4 Git and SVN

4.1 The polite programmer presents - Torvalds Vs the world of CVS

$$Git > Mercurial > tar + patches > SVN > CVS$$

Linus T. preferred source management methods are shown in relation 4.1.

4.1.1 History, Bitkeeper

An early SCM, decent but why settle for less than perfect. Git's flow is partially based on Bitkeeper but this was commercially distributed system which people did not like and could not even try to improve.

4.1.2 Future, what features would a replacement have?

1. What goes in, comes out. Undetected corruption is not an option.
2. Performance, if it takes too much time...
3. Merging easy, Branching is trivial. Merging should be easy.
4. Diff, to show what and how much changed. But why only on files?

4.1.3 A distributed system

What are possible motives for switching to a distributed system,

- Centralized collaboration is hard due to branch cost, more accurately merging branches.
- You dont́ want a central location waiting to fail or be compromised.
- You dont́ want any single point of failure.
- You can work offline for longer periods of time.
- By distributing you have a crude form of backup as all copies can be used equally.

However, there are some properties a distributed system has which reduce the fun of building, using and maintaining it, for example branching is required for any (offline) use. While the branching can be rather cheap in terms of storage and computational requirements, the demand for constant merging will require special care and introduce overhead in using the system. Using a distributed system you can benefit from a Trust network, by modeling the repositories to be compatible you can easily incorporated/pull code

and patches from those peers you trust, and by extension and smart merging, the peers they trust. Each person can determine who has access to their own repository by means of their own choice. A use case for this would be a release team that can read from a developers repository[2] and carefully test all changes they introduce into a 'verified' release grade repository.
A team of developers, one of potentially many, can share their branches and code without continuously running into conflicts with other teams. Therefore enabling them to develop complex experimental features collaboratively, while maintaining a stable codebase for other teams who might work on their possibly related or even conflicting features.
A decent distributed code management system will allow a user to track the source and history of code, features and bugs. By making sure a full history is available you can enjoy the power of quality reviews, or audits should someone expect foul play.

4.1.4 The powers of Git

- Even if you don't care about Git and just adore CVS[3] you can use Git just for merging, then put it all back in CVS.
- Git makes merging easier than the competition by automating what it can.
- Merging can be delegated to the best knowing party, in the case of a conflict the person who caused it will know best what to do.
- You don't need to see or share all existing branches, a developer work in a private experimental branch and push it out when done or decent.
- Branch names need not be unique across the globe so you can use natural names
- A 'super' project can contain references to multiple projects which can be shared, much like libraries of code.
- Git checksums everything, cryptographically secure, this to ensure the data you put in you also get out. It makes it harder to sabotage the history with nasty surprises.
- Trust your data to be valid, Git does all it can to detect corruption for you.
- Trust the data even from a shaky source, as long as you have the checksum from another more reliable source.

[2]but not write if they choose to
[3]Do not let L. Torvalds know this, for your own sake

- Git can track full history of changes, however if you really need to you can rewrite it, but you can't really hide the act.
- Tracks history regardless of the files the content lives in, tracking history even if you move a piece of text around.

4.2 Hg Vs Git

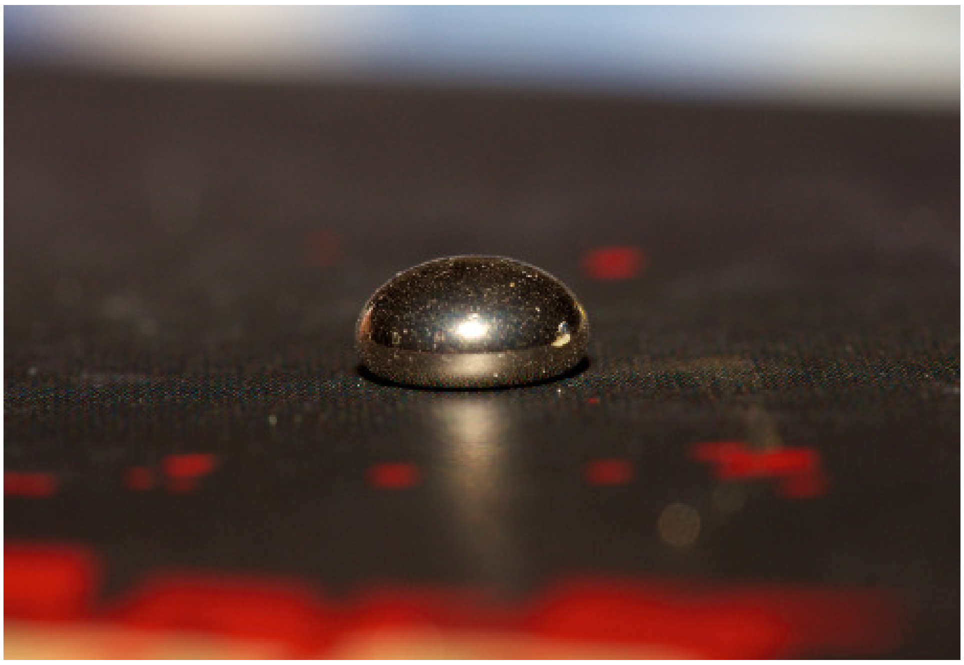

Figure 3: Hg Blob, the metal

Due to the nature of Mercury it would be dangerous to confuse the metal with the software, so illustrated in Figure 3 is the dangerous metal, courtesy of Flickr[4]. Do not try to use any object like the one shown for version management of any code. You have been warned.
When comparing Mercurial[5] with Git you can run into these statement, each relevant to selecting the system suited for you.

- Hg history is sacred, you cannot change it.
- Git lets you rebase/change/destroy history 'easily' if you explicitly instruct it to.
- Git does not really cater to SVN users, by design.

[4]http://www.flickr.com/photos/hyper7/9530010713/, Creative Commons license
[5]Also known as Hg

- There are plugins for both, with Hg having a tad more windows friendly software, and Linux being the most efficient platform for Git. If you use Eclipse just ignore the operating system, it has plugins for everything and anything.
- Hg cares about windows, which could be useful when you run into a bug or want the last bit of performance you want. Git can be run on windows at a small performance reduction when compared to Linux, but Git is more pleasant in a Linux environment.
- Hg might be more backwards compatible[6], though we do have this on authority.
- Hg can support shell written extensions using aliases, allowing core features to be accessed. This is a powerful way to write plugins to extend the power of Hg.
- There are always options to buy commercial support for Hg. Though Git will also have it's share of knowledgeable consultants[7].
- Hosting a git or Hg server can be done or done for you easily at affordable fees.

Why could you choose Git as the best system for You?

1. Things in git are immutable, being able to revert changes and undo mistakes is a useful safety net.
2. Git history is safe and stored indefinitely, unless you go to the trouble of telling Git to like in Section 4.2 Rewriting history can be done in both Hg and Git, the Git way of doing so being quite powerful and once you know how, easier.
3. Bragging rights, just because *Branch* sounds cooler than *bookmark*, and is actually a core feature.
4. Git allows partial files to be committed, leaving some changes local if wanted (Such as a 'comment to self, spellcheck my name'). This still holds even for larger files, where it becomes much more useful.
5. *git blame -C -s* Track ownership/history even if you move a feature or bug from a.file to b.file.

[6]Wikipdia style, Citation needed
[7]Citation desired

4.2.1 A simple comparison of VCS's

There is always one rule to manage by; Do not anger the programmer, choose the right DVCS

Feature	CVS	SVN	Git	Hg
Mature	+++	++	+	+
Track moved content	-	-	+++	+
Atomic operations	-	+	+	+
Branching possible	+	+	+	+
Branching pleasant	-	-	+++	+
P2P	-	-	+++	++
Speed	-	-	+++	-
Offline history	-	-	+	+
Windows compatible	+	+	+	+++
Learning curve	+	+	-	+
Documentation	+	+	+	+
2 parent merge	-	-	+	-
Extensions	-	-	-	+
Core power	-	-	+	-
Small hosting cost	low	low	low	low
Huge hosting cost	High	High	low	low

Of course, learning to use any new system is bound to slow down progress for a while, so why not consider keeping the system that has been in place for a while now? Are you running into any problems? Changing VCS will manage cause at least some problems.

5 LaTeX

5.1 Tex vs. Word vs. Writer

The article goes into detail about how LaTeX compares to MS Word en OO Writer. In my opinion, it is a bit short-sighted to only state Latex's typesetting features. Of course it has a bigger flexibility compared to the alternatives, but this doesn't mean that it outperforms its competitors qua usability. LaTeX offers extensive control over the display of the document and fonts. For instance, control over ligatures, and even control over the spacing between ellipsis dots. It is nice that you CAN do these things with LaTeX, but the situations in which I actually needed these features are limited to a handful. 99% of the day to day use doesn't need this extensive control and in these cases I would favor using WYSIWYG editors. If I have to make a report with a lot of mathematical equations I could imagine LaTeX being my tool of choice.

Word offers themeing and templating features that are available at the distance of a mouse click. If used correctly, it is a powerful tool. New users are inclined to apply their formatting manually though, which brings inconsistencies along.

– *Cedric*

Latex wins most friendly features, with Word and Writer in a tie. Latex has an edge with larger documents, being plain text input. Word and Writer have the power of integrating Spreadsheets and other 'documents' which can be quite useful.

A Latex document is like a program, and a Word/Writer document is easy to get 'good' output from, but very hard to get 'better than good' output. With all the control you have in Latex it might be a tad harder to get the output you want, but once there it tends to be magnificent.
A comparison is made in table 2.

– *Leendert*

5.2 Word vs. LaTeX

The second article compares MS Word and LaTeX also, but has a few errors in it: Word does have bibliography and citation features on board, although stated otherwise. Compatibility problems become less and less of an issue for MS Word, as it also supports the open document format nowadays, and its docx extension is basically a zipped version of an XML file with formatting applied to it. It is true that only experienced users know how to make good use of Word's layout features, but compared to the learning curve of LaTeX, I think the attributed stars to both cancel each other out. I have to agree though on the price and availability part.

– *Cedric*

Small documents ease, goes to Word. Setting the simple markup is just easy and tweakable. Once you have larger documents the Word markup can be good, but in general Latex markup is advanced and can be styled as you want it.

When handling larger documents Word can become unwieldy, slow. Latex only processes it when you are done, and can get something to eat. Not forcing you to wait during your moment of inspiration.

If you want a simple document word has the best learning curve, type and a few clicks get your result. Latex however requires basic knowledge of Latex prior to getting results. Once you have the knowledge most features can be used more efficiently than in Word's click model.

Cost of ownership is in favor of Latex, being free and open. This also goes for the easily integrated citation tools.

Storing the results in Latex is often PDF, which is widespread, and not dependent on one vendor like the default Word output.

In general, if you know neither, and want a short document grab Word. If you want a Long document, even if you don't know any Latex, investing the time will probably be worth it. And once you know the basics, just pick what suits you.

– *Leendert*

5.3 Why should you use Latex

> The guy on stackexchange sums it up quite well: you shouldn't use LaTeX if you don't have time to learn it, or if the design is more important than the typesetting of the document. I do like how the content of the document is separated from the formatting of the document. Although newer versions of Word also use XML documents for the content, while storing the formatting elsewhere.
>
> – *Cedric*

> It is portable, human readable file format (any text editor can be used to salvage a document, even if you don't have any Latex/related tools). Try doing that with word.
>
> You separate styling and content, keeping the content cleaner and styling reusable.
>
> – *Leendert*

5.4 Writing a thesis: Word or Latex

> The narrator felt frustrated about Word, because Word didn't do what he expected it to do. He says that LaTeX is good at what Word is bad at (and I agree). Although bibliographical features are available in Word, you have much more control over them in LaTeX.
>
> – *Cedric*

> Word is complex, hard, impish Latex has advanced automatic features, easily invoked.
>
> Style is Latex's thing, it does it good. Word does a lot for you, which it can decide to do whenever you don't want it. Saving, messing up intended layout and more.
>
> The method of writing differs, Latex takes getting used to. Borrow an example, learn Latex the easy way. It's worth it.
>
> – *Leendert*

Feature/support for	Word	Writer	LaTeX
Small caps	2	1	3
Proper numbers in text	1	1	3
Ligatures	2	1	3
Badness	1	1	3
Hyphenation	US	US	GB
Drop cap	2	2	3
...	1	1	3

Table 2: A comparison of LaTeX against the *others*

A References

References

[1] IBM. Introduction to XML: The Basics. `http://www.youtube.com/watch?v=Q0k5ySZGPBc`. Last accessed: 2013-09-13.

[2] IBM. Introduction to XML: XML and Web 2.0. `http://www.youtube.com/watch?v=16q5bbeO3xI`. Last accessed: 2013-09-13.

[3] IBM. Introduction to XML: Managing XML Data. `http://www.youtube.com/watch?v=R63g3h5f5oA`. Last accessed: 2013-09-13.

[4] David A. Wheeler. Introduction to the autotools, part 1. `http://www.youtube.com/watch?v=4q_inV9M_us`. Last accessed: 2013-09-13.

[5] David A. Wheeler. Introduction to the autotools, part 2. `http://www.youtube.com/watch?v=tku2h_fPxhc`. Last accessed: 2013-09-13.

[6] David A. Wheeler. Introduction to the autotools, part 3. `http://www.youtube.com/watch?v=e-uYBb554LU`. Last accessed: 2013-09-13.

[7] Peter Miller. Recursive make considered harmful. *AUUGN Journal of AUUG Inc*, 19(1):14–25, 1998.

B Contribution and authorship

Name	What	Reference
Cedric Van Bockhaven	Git setup	
Cedric Van Bockhaven	XML and XHTML	1
Cedric Van Bockhaven	LaTeX	5
Azad Kamali	Further Reading, XML and XSLT concepts	2
Esan Wit	Auto-Tools	3
Esan Wit	Minor rewriting XSLT concepts concerning XPath	2
Leendert van Duijn	Makefile	
Leendert van Duijn	Initial skeleton	
Leendert van Duijn	Git and SVN	4
Leendert van Duijn	LaTeX	5