Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di laurea in Informatica

PRINTING BOOKS FROM HTML AND CSS: METRICS, FORMATTERS AND RESULTS

Relatore:

Dott. ANGELO DI IORIO

Presentata da:

ANTONIO DI MARCO

I Sessione

2018/2019

Table of content

1 Introduction

Publishing a book is notoriously not an easy fit, involving of course writing the content but also a complex set of surrounding activities like proofreading and printing. In this text we will focus on the latter, which is propaedeutic to most of the other activities in particular on the quality of the produced books, which also affects their fruition once out of the shelves.

Publishers often have a set of non-negotiable requirements for their books, involving not only restraints derived from how the text in the book is visually presented to the end user (example of which could be font types, font styles, text alignment) but more complex ones dictated by how the printers bind the pages together and the fabric of cover and pages. Moreover, with the relatively recent explosion of handheld devices like mobiles and tablets as reading media, typographers have started looking with more and more interest for ways to generate ebooks that resemble the printed version of their books. In doing so, quite understandably, arises the need to minimise the differences in producing one or the other format and whenever possible, between the printed version and its digital counterpart.

One possible option to represent a book is to represent its content in an XML language like XHTML, and then to apply an XSLT transformation to it so to produce an XSL-FO (eXtensible Stylesheet Language - Formatting Objects) document. XSL-FO is “an XML vocabulary for specifying formatting semantics”[1] or, in other words, a markup language for formatting XML documents that is readable by a so called FO processor, that will in turn output the document in its final printable or readable form, for example a Portable Document Format (PDF).

[1] *The Extensible Stylesheet Language Family*. Available at https://www.w3.org/Style/XSL/ [last accessed 08 June 2019]

There are several XSL-FO processor available in the market, both open source like Apache FOP, and commercial like AntennaHouse XSL Formatter.[2]

In its latest version, XSL-FO 1.1 was published in 2006 by the XSL (Adler, 2000) Working Group and the group has since been closed[3] and no further development can thus be expected.

In parallel, CSS (Cascading StyleSheet) (Bos, 2011), one of the three technologies at the base of the World Wide Web development, has gained traction and in its new release it now features a module that "specifies how pages are generated and laid out to hold fragmented content in a paged presentation" which is targeted at printing media. As with XSL-FO, exist software, called *formatters*, that translates a combination of HTML and CSS to PDF and other output formats.

In this work, we aim at understanding whether constructing books with HTML and CSS3 constitutes a valid alternative to XSL-FO and in general as a viable option for publishers striving to produce a *camera-ready* artifact. There exists precedent work to analyze and benchmark CSS3 Paged Media, namely from W3C[4] and *Print CSS Rocks*[5] a website created to teach it by example and collect information about the formatters but, differently from these, that look at the single properties of CSS3 Paged Media, the present document focuses on the different perspective of analyzing the final result on real cases and on books in their entirety.

To achieve a valid result, we collected the requirements from an italian publisher (Casa Editrice *Il Mulino*) and tested whether CSS3 provided ways to comply with them. Additionally, we compared the results of several products, both from the open source

[2] See http://www.sagehill.net/docbookxsl/FOprocessors.html for a more extensive list of processors.

[3] *The Extensible Stylesheet Language Family*. Available at https://www.w3.org/Style/XSL/ [last accessed 08 June 2019]

[4] CSS Paged Media Module Level 3 CR Test Suite Results. http://test.csswg.org/harness/results/css-page-3_dev/grouped/

[5] *Print-CSS Rocks*. https://www.print-css.rocks/

offering (Weasyprint[6]) and the closed one (PrinceXML[7], AntennaHouse Formatter[8], PDFReactor[9]), evaluating the level of support of CSS3 for print.

The initial part of the work was dedicated at selecting what types of metrics and identifiers could portray objectively the status of CSS3 support for the case in use, starting with the already mentioned requirements from *Il Mulino*: as every publisher they convey a good part of their output review focus on the way the text is displayed and on the user experience. As an example of it, a renowned guide style as the *Chicago Manual of Style* (University of Chicago, 2010) states that “a page should not begin with the last line of a paragraph unless it is full measure and should not end with the first line of a new paragraph", respectively called widows and orphans. It’s common that different publishers will want to extend this definition to a custom number of lines. Widows and orphans, and the number of instances in a given document are examples of indicators and metrics that we considered in the test. More details and additional requirements will be presented later on.

It’s possible to categorize the properties of the PDFs in two main groups: the first containing properties that are common to all documents (e.g. widows and orphans), and the ones that denote a specific editorial collection (e.g. font family, font size, line spacing, etc.).

In the initial part of the work we used the latter, a selection of the *Il Mulino'*s requirements, each of them represented in a separate PDF that shows the intended behavior (see chapter 4.1.1) to compare the different CSS “formatters”, the software that combines HTML and CSS, and produces a PDF.

---

[6] *Weasyprint* - https://weasyprint.org/

[7] *PrinceXML* - https://www.princexml.com/

[8] *AntennaHouse Formatter v6*. https://www.antennahouse.com/formatter/

[9] *PDFReactor* - https://www.pdfreactor.com/

For the second set of tests we used PDFs representing entire books, such as “I promessi sposi” by Alessandro Manzoni (see chapter 4.1.2 for a description of this test group and the full list of books under inspection), and produced a software that automatically scans the PDFs and checks against a list of previously identified indicators of quality (e.g. presence of widows or orphans above a predefined threshold). To do so we used *pdfminer*[10], a python library that is able to parse a PDF into a browseable data structure. As per the formatters, a number of library to scan PDFs exist and have different levels of quality. A comparison of PDF parser libraries constitutes enough work for a separate research.

Lastly, using the experience from previous research and other academic studies, we came up with additional test indicators and generated a combination of HTML and CSS to test both the CSS support and the formatters behavior (see chapter 4.1.3).

Conclusions summary

Each formatter has different level of support of the various CSS directives and even when supported, the visual disposition of elements differ noticeably.

CSS supports the implementation of the requirements we investigated (widows, orphans, keep properties) exhaustively. The production of PDFs is yet not entirely automatable and manual intervention is necessary specifically when e.g. space between two blocks or items depends on the context (space conditionality) or when we want to insert a blank page. Even though there are known gaps on edge cases, it was possible to cover all *Il Mulino* visual requirements and of text disposition, simulating correctly the books we were given as test base. The tested formatters produced different levels of quality, analyzed further down in the document. Among them, PDFReactor is the one that creates the least amount of rules violations, while producing an output that is the easiest to parse correctly with PDFMiner. As the result of this research, we can state that it is definitely possible to produce a PDF with enough

[10] *PDFMiner*. https://github.com/euske/pdfminer

professional quality to be considered as the final work for printing books. In comparison to FO processors, the CSS formatters are less sophisticated so that small manual adjustments to the HTML/CSS were needed. Most of the commercial formatters created a proprietary set of CSS properties[11] to simplify or enhance the job.

Document overview

Chapter 2 explores the background and related works in the space of text representation.

Chapter 3 lays down the indicators and properties with which we'll evaluate formatters.

Chapter 4 describes the experiment.

Chapter 5 draws the conclusions of the work.

[11] Example of this being *Page groups* by PDFReactor.
https://www.pdfreactor.com/product/doc_html/index.html#PageGroups

2 Background and related work

In this chapter we discuss XSL-FO and CSS, two of the most common languages used to represent documents and their formatting, and then explore the background information, protocols, languages and research that precede the current work.

2.1 XSL-FO

Copious work has been directed at formalize a way to generate accurate content with formal and precise instructions, with great results like in the case of LaTeX (Knuth, 1983) which is out of scope for this work.

The most important effort in the world of XML, is XSL-FO (Berglund, A. *2006*) (eXtensible Stylesheet Language), "a technology for creating paginated print versions of information contained in XML documents"[12].

An XSL-FO document is an XML document that follows the schema under the *fo* namespace[13]. Starting from the XML document, designers transform it into XSL-FO using a transformation written in XSLT and then use an XSL-FO formatter to generate the final document in PDF, postscript, or one of the other supported formats. The XSL document's content constitutes a tree of elements, or formatting objects, each one of which, represents an output area, whose content is specified in the XSL document itself. Areas have predefined positions in the page (see Figure 2.1a) and a set of properties[14].

[12] *XSL-FO*, Dave Pawson, O'Reilly (2002*)*

[13] *Fo namespace*, http://www.w3 .org/1999/XSL/Format

[14] *XSL-FO*, Dave Pawson, Chapter 4. O'Reilly (2002*)*

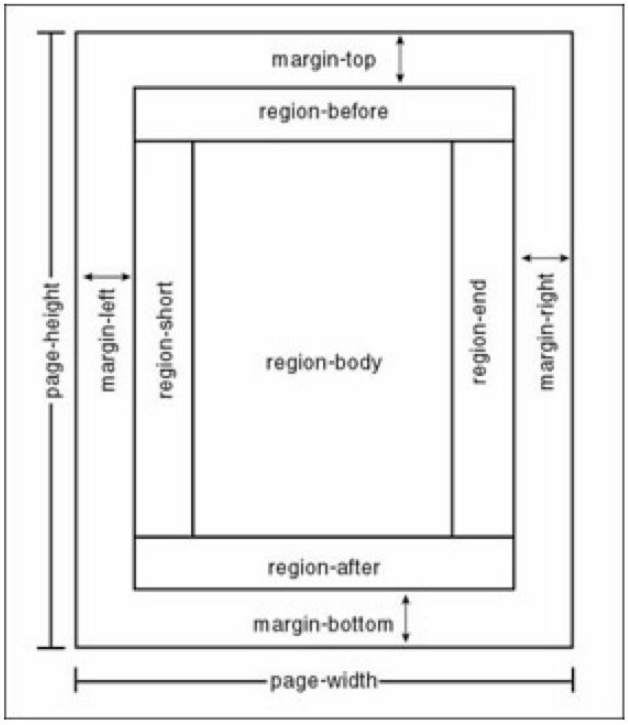


Figure 2.1a - An area as represented by XSL-FO

One of the most important novelties introduced by XSL-FO is the concept of *flows*, a way to represent a sequence of pages in the XSL tree. *Flows* are of two types: <fo:flow> flows, for content that spans across multiple pages, and <fo:static-content>, for content that is repeated in multiple pages (e.g. footers). As we'll see later in this document, CSS doesn't have the support for flows or a similar concept to address multiple pages at the same time.

Among the many options for fine tuning, XSL-FO allows for the management of the minimum number of lines to appear alone at the bottom of the page (so called *orphans*) and at the beginning of a page (*widows*) as the result of a page break.

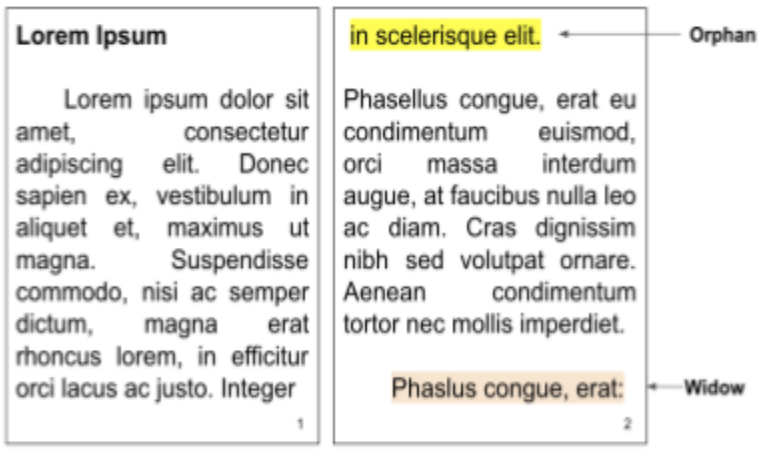


Figure 2.1b - In evidence, an *orphan* and a *widow*.

As an example, the code to set the orphans inside an *fo:block* is as follows:

```
<fo:block widows="3" orphans="3">Paragraph content</fo:block>
```

Which the formatter will apply to the document as shown in Figure 2.1c.

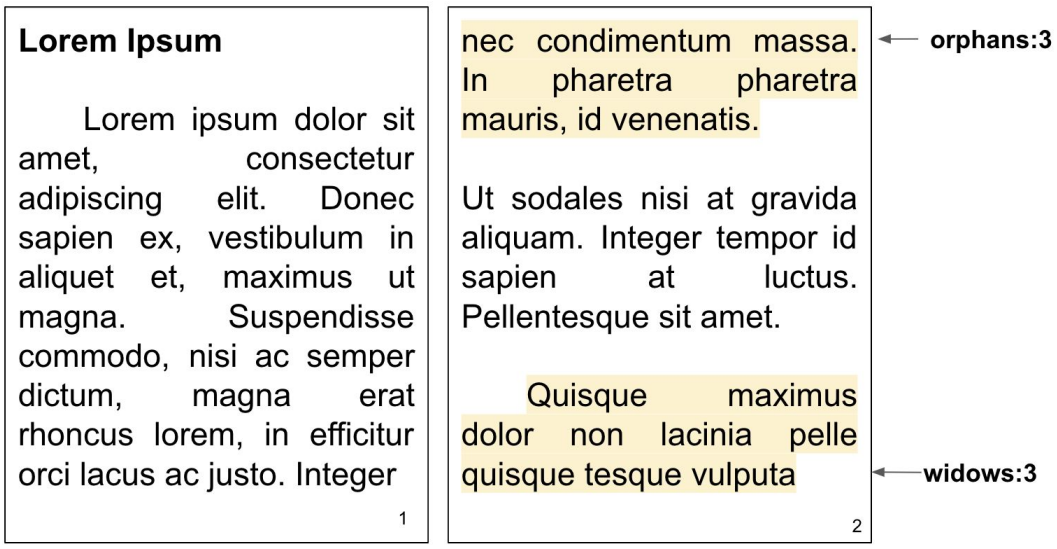


Figure 2.1c - Applied setting of maximum widows/orphans lines after the processing

Similarly, page breaks within an fo:block can be forced (see Figure 2.1d) through

```
<fo:block page-break-before="always">
```

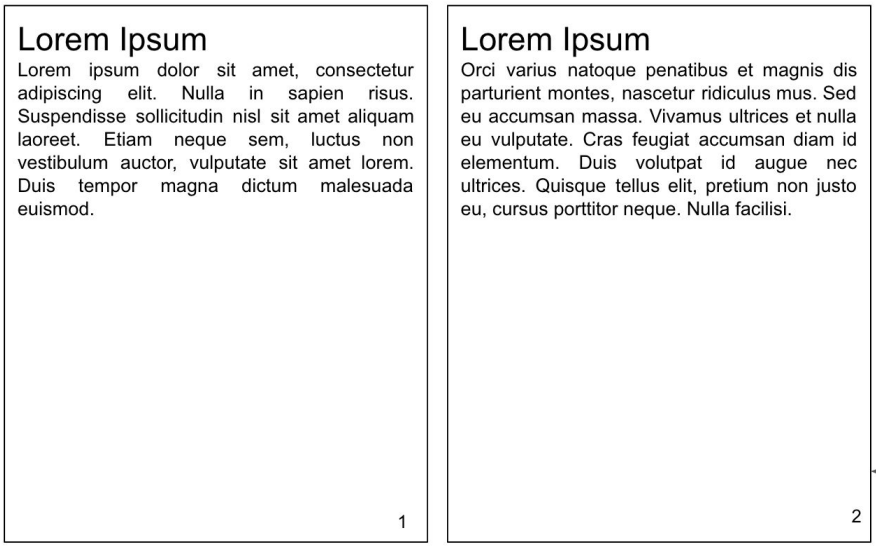


Figure 2.1d - A page break is artificially inserted before the start of the second chapter.

2.1.1 Status of the standard

This standard is designed for completeness but not for ease of use, making it impractical to manually edit. Its limited capabilities especially in respect to conditional and relative formatting is one of the reasons for the lack of adoption by commercial XSL-FO implementers. Concurrently, the rise of a simpler language like CSS, the pressure on CSS and on browser vendors from publishers and the reduced participation to the w2c working group meant in practice the demise of the working draft.

2.1.2 XSL-FO processors

After transforming the XML file containing the text of the book to adhere to the XSL-FO schema, the last step is to produce the final file (PDF, postscript, etc.). This is the role of the so called *XSL-FO processors*.

During the years, several processors appeared on the market, as often happens in both commercial and open source licensed. The most famous were Apache FOP[15], RenderX Xep[16], AntennaHouse Formatter[17] and PrinceXML[18], the last two of this list now expanded their supported inputs also to CSS3.

2.2 CSS

CSS is a language for describing the rendering of structured documents (such as HTML and XML) on screen, on paper, etc. When the first CSS specification was published, all of CSS was contained in one document that defined CSS Level 1. CSS Level 2 was defined also by a single, multi-chapter document. However for CSS beyond Level 2, the CSS Working Group chose to adopt a modular approach, where each module defines a part of CSS, rather than to define a single monolithic specification. This breaks the specification into more manageable chunks and allows more immediate, incremental improvement to CSS4 (Etemad, 2018).

Among the modules defined in the working draft for CSS3, the most relevant for this work is the one called CSS Paged Media which specifies how pages are generated and laid out to hold fragmented content in a paged presentation. It adds functionality for controlling page margins, page size and orientation, and headers and footers, and extends generated content to enable page numbering and running headers / footers.

While CSS Paged media deals with the layout of elements in a document, a separate module called CSS Fragmentation covers how to instruct CSS to split content flow into pages, columns or regions.

---

[15] *Apache FOP*. https://xmlgraphics.apache.org/fop/

[16] *RenderX tools index*. http://www.renderx.com/tools/index.html

[17] *AntennaHouse Formatter*. https://www.antennahouse.com/formatter/

[18] *PrinceXML*. https://www.princexml.com/

In CSS, each page is composed by sixteen page margin boxes that can display generated content like a pseudoelement (Figure 2.2a)

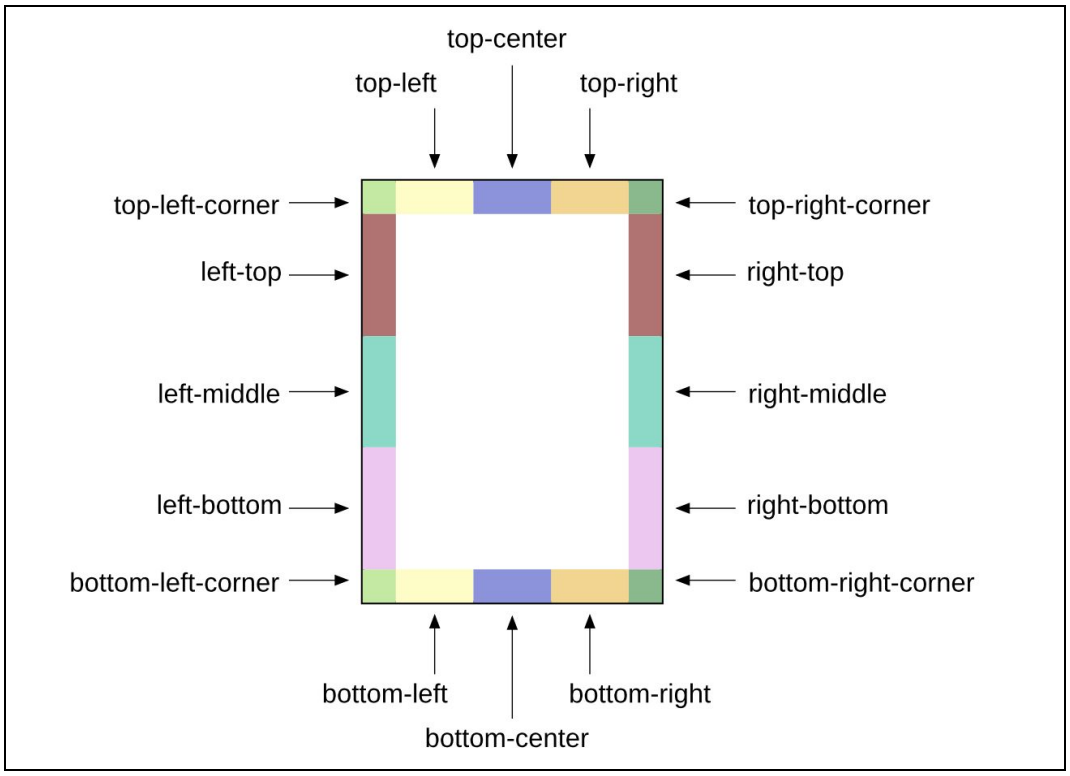


Figure 2.2a - Page margin boxes (pdfreactor.com)

The at-page (@page) rule is used to control the page characteristics when printing. With this rule, CSS makes it possible to change the margins, orphans, widows, and page breaks of the document.

The target on printing is obvious when thinking that, for example, size supports not only the classic measurement units like inches but also absolute sizes like A4.

Additional flexibility is given by the pseudo-page selector, that allows to target different behavior for the pages on the right or left, as well as the first page and blank pages.

```
@page :first {
  size: A4;
  margin: 1cm;
}
```

Inside the at-page block, it's possible to target the various areas displayed in Figure 2.2a, namely @top-left-corner, @top-left, @top-center, @top-right, @top-right-corner, @bottom-left-corner, @bottom-left, @bottom-center, @bottom-right, @bottom-right-corner, @left-top, @left-middle, @left-bottom, @right-top, @right-middle and @right-bottom.

Additionally to the above described areas, it's possible to describe the behavior of footnotes inside an at-page selector. When applying "float: footnote" to an element, it becomes a footnote, which means it's removed from the flow and placed at the bottom of the page. Some more actions are auxiliary to this event happening: the footnote counter is incremented and the footnote marker is placed at the beginning of the footnote.

In code, a footnote can be created as follows:

HTML

```
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec
venenatis, dui id molestie varius, sem velit dignissim ligula, eu
varius leo sem non ipsum. Nunc pretium eu neque non elementum.<span
class="footnote">Integer porttitor fringilla leo ac
elementum.</span></p>
```

CSS

```
@page {
  @footnote {
    float: bottom;
  }
}

span.footnote { float: footnote; }
```

The code results in the PDF in Figure 2.2b.

Figure 2.2b - PDF containing text and footnote

Widows and **orphans** are controllable similarly to XSL-FO, with instructions embedded directly into the language itself:

```
/* <integer> values */
widows|orphans: 2;

/* Global values */
widows|orphans: inherit;
widows|orphans: initial;
widows|orphans: unset;
```

Also common with XSL-FO is the management of page-breaks and keep properties.

To keep two boxes together in the same page, CSS provides the instructions to avoid or force the page break:

```
break-after[19]

break-before[20]
```

as well as other instructions intended for break management at the level of the page[21]

```
page-break-before[22]

page-break-after[23]
```

the box

```
box-decoration-break
```

and inside the element itself

[19] *Break-after.* https://developer.mozilla.org/en-US/docs/Web/CSS/break-after

[20] *Break-before*. https://developer.mozilla.org/en-US/docs/Web/CSS/break-before

[21] replaced in the newest versions of CSS by break-before and break-after that we report here for completeness

[22] *Page-break-after.* https://developer.mozilla.org/en-US/docs/Web/CSS/page-break-after

[23] *Page-break-before*. https://developer.mozilla.org/en-US/docs/Web/CSS/page-break-before

```
break-inside
```

The break-after and break-before rules work on different levels (generic, page, column and region), depending on where they are applied:

```
/* Generic break values */
auto, avoid, always, all

/* Page break values - inside @page */
avoid-page, page, left, right, recto, verso

/* Column break values */
avoid-column, column

/* Region break values */
avoid-region, region

/* Global values */
inherit, initial, unset
```

For example an instruction of “break-before: left;” will force “*one or two page breaks right after the principal box, whichever will make the next page into a left page*”. On another level, “break-before: avoid;” will keep two boxes together in the same page, whenever possible.

Unlike XSL-FO, there is no concept of flow or page sequences, so that the behavior of multiple pages must be instructed individually.

2.2.1 CSS formatters

A good number of closed source, commercially licensed software is present on the market to aid the typographers achieve a perfect printed or digital result, but the price of said tools is quite high (see table 2.2.1a for a price comparison among the most famous ones) given the limited size of this niche market and the elevated number of different requirements that the software needs to comply with, to satisfy the users.

Formatter	**License**	**Price**
Weasyprint	OpenSource	Free
AntennaHouse[24]	Proprietary	5000$
PrinceXML[25]	Proprietary	3800$
PDFReactor[26]	Proprietary	2680$
WKHTMLtoPDF	OpenSource	Free

Table 2.2.1a - List of formatters under analysis. WKHTMLtoPDF was discarded from the test for the lack of support of CSS Paged media[27]

Browsers and open source formatters

Open source alternatives to the commercial software are available and plenty, but the quality of the end product is not satisfying enough. Many of the tools that we tried, use an open source browser engine behind the scene (Chrome, Firefox, etc.) and those do

[24] *AntennaHouse*. https://www.antennahouse.com/prices/

[25] *PrinceXML*. https://www.princexml.com/purchase/

[26] *PDFReactor*. https://www.pdfreactor.com/buy/

[27] *WKHtmlToPDF*. https://github.com/wkhtmltopdf/wkhtmltopdf/issues/2066

not support the CSS3 paged media module in its entirety[28]. The quality of these software can't be considered good enough to be used in a professional environment, exactly for the lack of complete support that reduces the maneuver space that publisher have to fine tune the input page and reach their desired results.

2.3 Some academic studies on typesetting

The work done in this document is preceded, inspired and extension to a bulk of scientific work that explores several aspects of typesetting.

The most simple approach in typesetting is to fit as many words as possible in a line, with a greedy approach. According to Sneep (Sneep, 2005), this is the default for most browsers and word processors at the time of the analysis. An improvement over this basic algorithm can be found in the foundational article about breaking paragraphs into lines (Knut, 1981) used in LaTex. As per Knut optimal solution, penalties are assigned to words and lines to calculate the fitness of a line break at any given point and the one with less penalties is then used.

Starting from Knut's seeding work, further optimizations moved the focus from the line breaking to a more holistic approach, with algorithms that concentrate on the idea of multiparagraph (Ciancarini, 2012) and a global optimization of the page rather than the greedy approach for each paragraph.

Another take on global pagination can be find in Mittelbach (2019), which at the 16th ACM Symposium on Document Engineering presented a framework for an algorithm for page breaking that leverages the ideas of Knuth/Plass but focuses instead on the global layout rather than at paragraph level. The algorithm presented can be directly utilized in any TeX installation.

[28] *CSS Paged Media benchmark*. https://caniuse.com/#feat=css-paged-media

This last example shows how extensive research has been performed to expand the work of Knuth/Plass and how necessary is to keep investigating the scope of automatic typesetting.

2.4 Other benchmarks

Corollarily to the specifications of CSS3, the W3C Working Group maintains a test suite that, for each CSS property, checks if it's supported by the most common browser engines (AHFormatter, Blink, Edge, Gecko, Presto, Prince, Trident, WebKit and WebToPDF). The results are available on the *CSS Paged Media Module Level 3 CR Test Suite Results*[29].

An extensive source of information lies in Print CSS Rocks (Jung A., 2019), a website that aspires at being a one-stop shop for learning CSS Paged Media. Each lesson on the site focuses on a different CSS property and, along with it, offers a snapshot of the support level of three PDF formatters (PDFreactor, PrinceXML and Antennahouse)[30].

The first group of twenty basic lessons/benchmarks in the *Intro* category, targeting isolated CSS properties like footnotes, hyphenation, tables, etc. shows how the three formatters supports fully the standard. This group of tests, as well as the rest, are executed manually, i.e. generating the PDF and then comparing them visually.

The following thirteen *Advanced* tests, focus on rarer properties like footnotes on multiple columns, text with ligatures and pseudo-selectors like ::first-line. AntennaHouse and PrinceXML both respond well to the tests while PDFReactors fails three of them by not supporting the properties and showing an unmodified text.

The last general category contains twelve tests, more exotic in nature and with the intention of stressing the formatters. The support of a complex CSS example with CSS

[29] CSS Paged Media Module Level 3 CR Test Suite Results. http://test.csswg.org/harness/results/css-page-3_dev/grouped/. Retrieved on June 2019.
[30] *Print-CSS Rocks, Lessons*. https://print-css.rocks/lessons. Retrieved on June 2019.

positioning, fonts and some typography reports green on the three formatters while the coverage of third parties charting JS libraries is disappointing.

It is the opinion of the author that PDFReactor is what fits best their use case for professional usage.

While the tests of the W3C benchmark and the print-css rocks website are targeted at each individual CSS property, in this work we take a more holistic approach on the final PDF document in its entirety.

3 Evaluating CSS and CSS formatters

XSL-FO has been used to produce printable books successfully but the complexity in manual editing and the demise of its working group at W3C might signal its soon decline.

The publication of CSS3 and of its Paged Media module might constitute a good replacement for XSL-FO and it's the scope of this work to give an overview of the compatibility of CSS3 to do just that.

In this chapter we introduce three groups of indicators, capabilities present in XSL-FO and in all requirements from publishers: style properties that look at how text is laid out in the pages; content properties, that takes a closer look at text quality; keep properties used to check whether CSS is able to keep different logical groups of text together. This initial set of indicators is based on the analysis of existing books and on previous work on the subject (see Chapter 3.1). Our list (see table 3a) is not definitive and set the course for additional research that will be performed in the future to extend the test base and the set of indicators.

Stype properties	Content properties	Keep properties
Chapter titles at top of page (S1)	Page never ends with a colon (C1)	Title and body are kept on the same page (K1)
Chapter titles at odd pages (S2)		Long title is kept on the same page (K2)
Orphans not present (S3)		Title, subtitle and paragraph are kept on the same page (K3)
Widows not present (S4)		Images and their captions are kept on the same page (K4)
Too much space at the end of the page (S5)		Text in the same page (K5)
Too much space between text and notes (S6)		

Table 3.1 - List of indicators by property type

3.1 Style properties

Style properties are characteristics of the way the text is laid out on the page to improve the user experience while reading books. Some of those properties are obvious also because they are widely adopted, like the case of chapters starting on a new page that is on the right side of the book and its title is displayed at the top of the page. Some others like widows and orphans are more subtle and both their definition and strictness in application depends on the style choices of the publisher.

Chapter titles at top of page (S1)

It is normal for publishers to require chapters to start after a page break, so chapters titles are always found at the top of the page. The opposite is less frequent (see Figure 3.1a).

It's not a case that among the set of instructions included in both XSL-FO and CSS there is the capability to control the text at page level.

In CSS we find two specific directives related to the dispositions of chapters in pages:

1. The ability to create a page-break after a paragraph ends, or before a chapter starts. (See details of break-after and break-before later in Chapter 4.2.2)
2. The ability to make a page behave as a left or right page, which would occasionally create a blank page before, should it be on the wrong side of the book (See details of break-before later in Chapter 4.2.2).

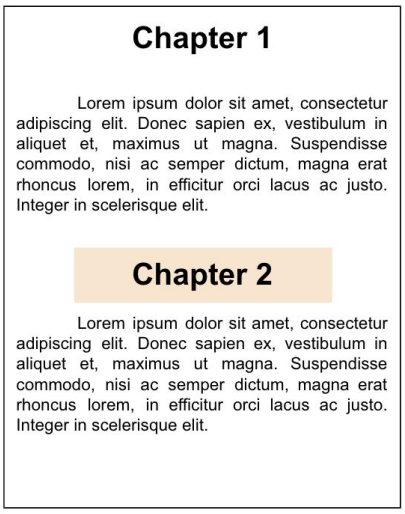


Figure 3.1a - Chapter title not at the top of the page

Chapter titles at odd pages (S2)

For ease of reading and better displaying, typographers use to print beginning of chapters on an odd page number. To force this, it's common practice to introduce a blank page at the left of a chapter, so to make it start at an odd page.

Orphans not present (S3)

In typesetting, orphans are lines at the end of a paragraph, which are left dangling at the top of a column, separated from the rest of the paragraph. There are different opinions on how many lines alone constitutes an orphan (Carter, R.,1993) (Day, B., 1993). Our requirements from *Il Mulino* stated that 3 is the minimum number. See Figure 4.1.a

Widows not present (S4)

In typesetting, widows are lines at the beginning of a paragraph, which are left dangling at the bottom of a column, separated from the rest of the paragraph. There are different opinions on how many lines alone constitutes a widow. Our requirements from *Il Mulino* stated that 4 is the minimum number. See Figure 3.1.b

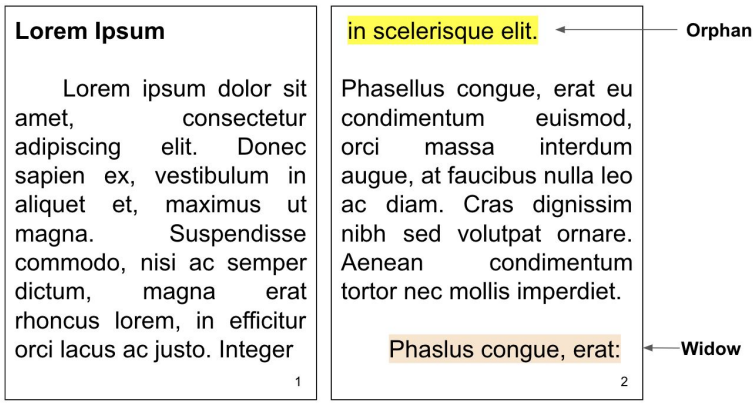


Figure3.1b - In evidence, an *orphan* and a *widow. The page ends with a colon.*

Too much space at the end of the page (S5)

For a better visual experience, page must be filled to the fullest. For the scope of this document and our tests, we considered 350pt as the maximum space allowed (Figure 3.1.c). This value is specific to the editorial collection in exam; different publishers might have their own preference.

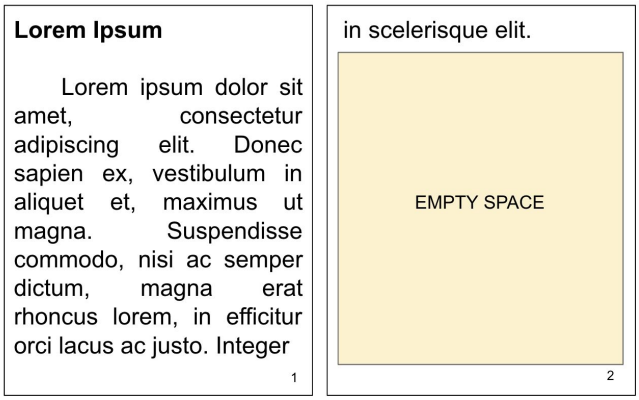


Figure 3.1.c - In evidence, large empty space at the end of a page

Too much space between text and notes (S6)

For a better visual experience, page must be filled to the fullest. When too much whitespace is present between text and the notes in a page, the page is not considered print worthy (Figure 3.1d)

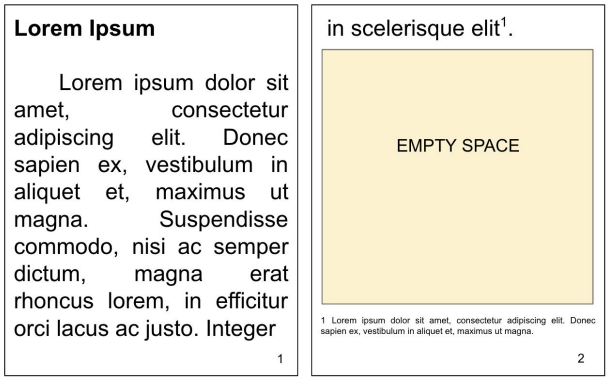


Figure 3.1d - Large empty space between the body of the text and the notes

3.2 Content properties

The properties in this category reflect the editorial choices regarding the style of the content. It stands out from the others because it doesn't observe the text layout but instead it refers to the meaning of the text. So far we identified one property, due to be revisited and expanded with additional items in future works.

Page never ends with a colon (C1)

Since a colon "precedes an explanation or an enumeration, or list", having a phrase ending at the bottom of the page to continue with e.g. a list on the next, interrupts the reading flow. See Figure 3.1.b

3.3 Keep properties

Properties in this category refers to the way the text is split among the pages of the book. The limited amount of space in each page forces the formatters to insert page breaks in the middle of chapters and sub-section, while the keep tags, both present in XSL-FO and CSS give enough flexibility to instruct otherwise and specify logical connections inside a block of text or even between different groups. An example could be the will of maintaining the title of the chapter and the initial part of its content in the same page (K1).

Title and body are kept on the same page (K1)

An HTML file that produces a PDF where, without any additional CSS instructions, a chapter subtitle appears at the bottom of the page, detached from the content of the chapter itself (Figure 3.3a)

Inserting break-after: avoid; in the "titoletto" class, the title and the body of the chapter are kept together. (Figure 3.3b)

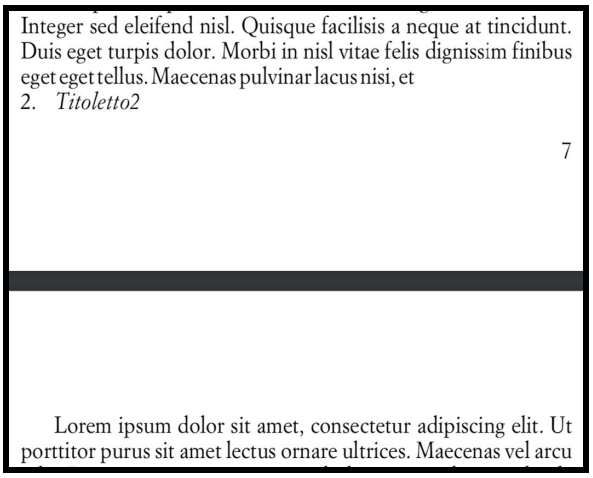

Integer sed eleifend nisl. Quisque facilisis a neque at tincidunt. Duis eget turpis dolor. Morbi in nisl vitae felis dignissim finibus eget eget tellus. Maecenas pulvinar lacus nisi, et
2. *Titoletto2*
7
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut porttitor purus sit amet lectus ornare ultrices. Maecenas vel arcu

Figure 3.3a - Title detached from body

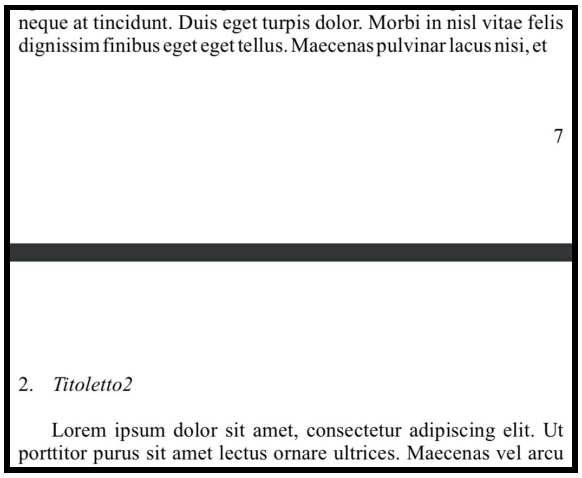

neque at tincidunt. Duis eget turpis dolor. Morbi in nisl vitae felis dignissim finibus eget eget tellus. Maecenas pulvinar lacus nisi, et
7
2. *Titoletto2*
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut porttitor purus sit amet lectus ornare ultrices. Maecenas vel arcu

Figure 3.3b - Keep property in action

Long title is kept on the same page (K2)

It is interesting to check what's the default behavior of the formatters in treating a single line title who appears at the bottom of the page (Figure 3.3c), when it's extended to take more than one line. In this case, the formatter takes the most logical decision to keep the title together and move it to next page (Figure 3.3d). The behavior of breaking the title in two pages has never come up, in fact, there is no CSS instruction to produce it[31].

[31] *Break-inside*, https://developer.mozilla.org/en-US/docs/Web/CSS/break-inside

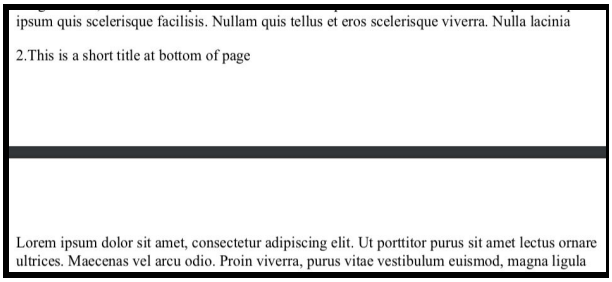


Figure 3.3c - a short title detached from body

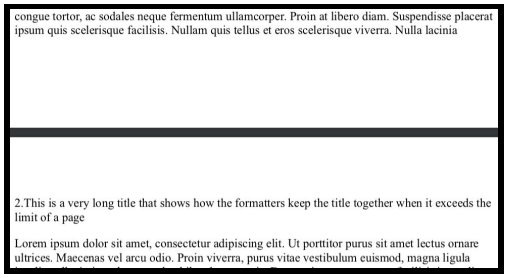


Figure 3.3d - A long title kept in the same page

Title, subtitle and paragraph are kept on the same page (K3)

All the headings preceding a text are best if present on the same page and followed by the content of that chapter. By default, formatters do not enforce this property that needs to be set manually. In the pictures it's possible to see the correct work of a formatter when the break avoidance setting with the following element is given (Figure 3.3e, f, g, h)

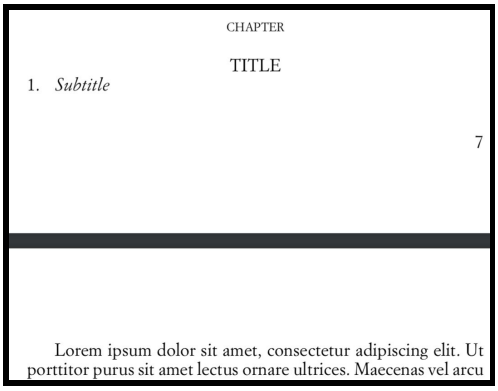


Figure 3.3e - Titles separated from body

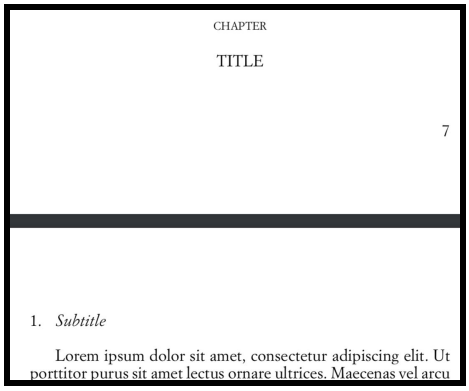


Figure 3.3f - First keep property, body

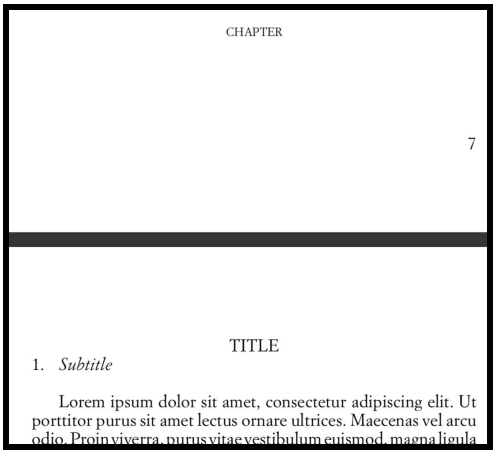


Figure 3.3g - Title and subtitle together

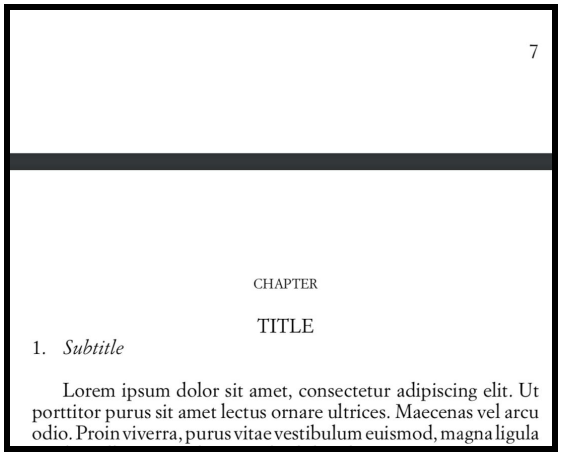


Figure 3.3h - All titles kept in the same page with the body

Images and their captions are kept on the same page (K4)

What is true for titles and the content of the page, is true for images and their captions. As visible in figure 3.1i, the default behavior of formatters is to separate the two elements that must be kept together with the right CSS setting (Figure 3.1j).

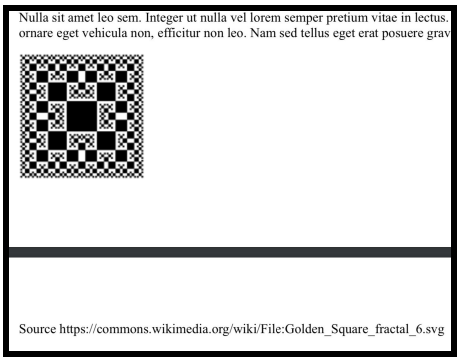


Figure 3.1i - Image and its caption are erroneously separated by a page break

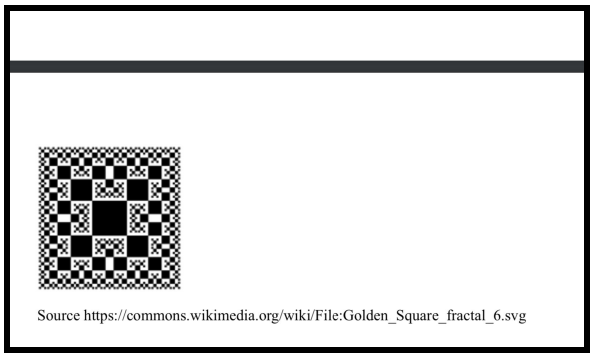
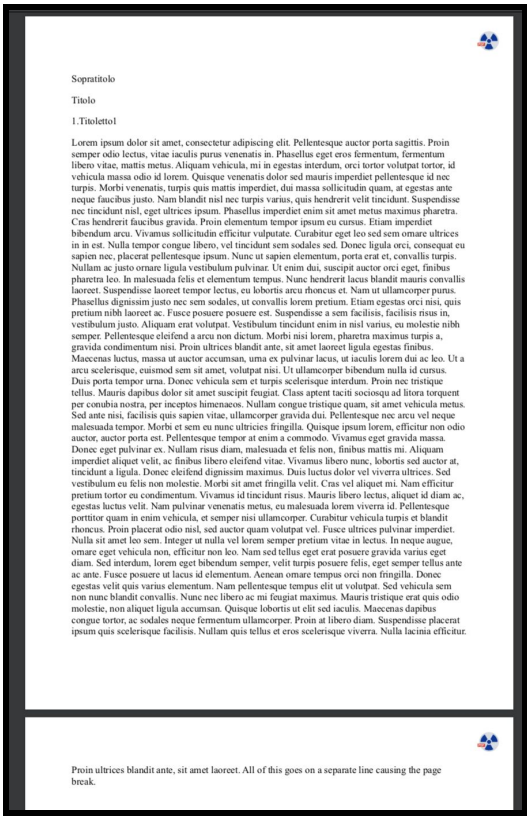


Figure 3.1j - Image and its caption are kept together by CSS

Text is kept in the same page (K5)

To avoid widows as the one visible in Figure 3.1k , one option is to instruct CSS to subjugate the text within the same page. The formatters have various options, e.g. reducing the line space, reducing the font, decrement the hyphenation at the expenses of line width or to ignore the setting for the space at the bottom of the page (Figure 3.1l)



Sopratitolo

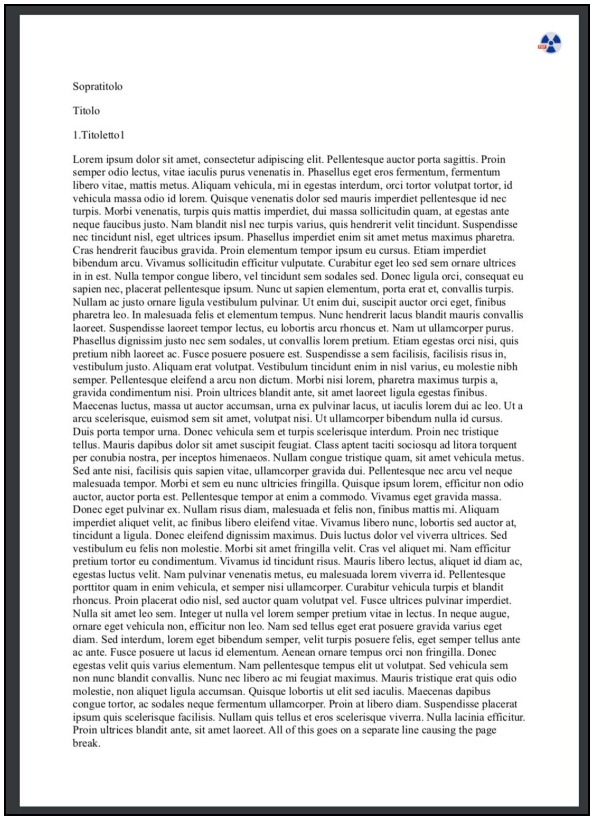
Titolo

1.Titoletto1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque auctor porta sagittis. Proin semper odio lectus, vitae iaculis purus venenatis in. Phasellus eget eros fermentum, fermentum libero vitae, mattis metus. Aliquam vehicula, mi in egestas interdum, orci tortor volutpat tortor, id vehicula massa odio id lorem. Quisque venenatis dolor sed mauris imperdiet pellentesque id nec turpis. Morbi venenatis, turpis quis mattis imperdiet, dui massa sollicitudin quam, at egestas ante neque faucibus justo. Nam blandit nisl nec turpis varius, quis hendrerit velit tincidunt. Suspendisse nec tincidunt nisl, eget ultrices ipsum. Phasellus imperdiet enim sit amet metus maximus pharetra. Cras hendrerit faucibus gravida. Proin elementum tempor ipsum eu cursus. Etiam imperdiet bibendum arcu. Vivamus sollicitudin efficitur vulputate. Curabitur eget leo sed sem ornare ultrices in in est. Nulla tempor congue libero, vel tincidunt sem sodales sed. Donec ligula orci, consequat eu sapien nec, placerat pellentesque ipsum. Nunc ut sapien elementum, porta erat et, convallis turpis. Nullam ac justo ornare ligula vestibulum pulvinar. Ut enim dui, suscipit auctor orci eget, finibus pharetra leo. In malesuada felis et elementum tempus. Nunc hendrerit lacus blandit mauris convallis laoreet. Suspendisse laoreet tempor lectus, eu lobortis arcu rhoncus et. Nam ut ullamcorper purus. Phasellus dignissim justo nec sem sodales, ut convallis lorem pretium. Etiam egestas orci nisi, quis pretium nibh laoreet ac. Fusce posuere posuere est. Suspendisse a sem facilisis, facilisis risus in, vestibulum justo. Aliquam erat volutpat. Vestibulum tincidunt enim in nisl varius, eu molestie nibh semper. Pellentesque eleifend a arcu non dictum. Morbi nisi lorem, pharetra maximus turpis a, gravida condimentum nisi. Proin ultrices blandit ante, sit amet laoreet ligula egestas finibus. Maecenas luctus, massa ut auctor accumsan, urna ex pulvinar lacus, ut iaculis lorem dui ac leo. Ut a arcu scelerisque, euismod sem sit amet, volutpat nisi. Ut ullamcorper bibendum nulla id cursus. Duis porta tempor urna. Donec vehicula sem et turpis scelerisque interdum. Proin nec tristique tellus. Mauris dapibus dolor sit amet suscipit feugiat. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nullam congue tristique quam, sit amet vehicula metus. Sed ante nisi, facilisis quis sapien vitae, ullamcorper gravida dui. Pellentesque nec arcu vel neque malesuada tempor. Morbi et sem eu nunc ultricies fringilla. Quisque ipsum lorem, efficitur non odio auctor, auctor porta est. Pellentesque tempor at enim a commodo. Vivamus eget gravida massa. Donec eget pulvinar ex. Nullam risus diam, malesuada et felis non, finibus mattis mi. Aliquam imperdiet aliquet velit, ac finibus libero eleifend vitae. Vivamus libero nunc, lobortis sed auctor at, tincidunt a ligula. Donec eleifend dignissim maximus. Duis luctus dolor vel viverra ultrices. Sed vestibulum eu felis non molestie. Morbi sit amet fringilla velit. Cras vel aliquet mi. Nam efficitur pretium tortor eu condimentum. Vivamus id tincidunt risus. Mauris libero lectus, aliquet id diam ac, egestas luctus velit. Nam pulvinar venenatis metus, eu malesuada lorem viverra id. Pellentesque porttitor quam in enim vehicula, et semper nisi ullamcorper. Curabitur vehicula turpis et blandit rhoncus. Proin placerat odio nisl, sed auctor quam volutpat vel. Fusce ultrices pulvinar imperdiet. Nulla sit amet leo sem. Integer ut nulla vel lorem semper pretium vitae in lectus. In neque augue, ornare eget vehicula non, efficitur non leo. Nam sed tellus eget erat posuere gravida varius eget diam. Sed interdum, lorem eget bibendum semper, velit turpis posuere felis, eget semper tellus ante ac ante. Fusce posuere ut lacus id elementum. Aenean ornare tempus orci non fringilla. Donec egestas velit quis varius elementum. Nam pellentesque tempus elit ut volutpat. Sed vehicula sem non nunc blandit convallis. Nunc nec libero ac mi feugiat maximus. Mauris tristique erat quis odio molestie, non aliquet ligula accumsan. Quisque lobortis ut elit sed iaculis. Maecenas dapibus congue tortor, ac sodales neque fermentum ullamcorper. Proin at libero diam. Suspendisse placerat ipsum quis scelerisque facilisis. Nullam quis tellus et eros scelerisque viverra. Nulla lacinia efficitur.

Proin ultrices blandit ante, sit amet laoreet. All of this goes on a separate line causing the page break.

Figure 3.1k - Text without a keep property



Sopratitolo

Titolo

1.Titoletto1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque auctor porta sagittis. Proin semper odio lectus, vitae iaculis purus venenatis in. Phasellus eget eros fermentum, fermentum libero vitae, mattis metus. Aliquam vehicula, mi in egestas interdum, orci tortor volutpat tortor, id vehicula massa odio id lorem. Quisque venenatis dolor sed mauris imperdiet pellentesque id nec turpis. Morbi venenatis, turpis quis mattis imperdiet, dui massa sollicitudin quam, at egestas ante neque faucibus justo. Nam blandit nisl nec turpis varius, quis hendrerit velit tincidunt. Suspendisse nec tincidunt nisl, eget ultrices ipsum. Phasellus imperdiet enim sit amet metus maximus pharetra. Cras hendrerit faucibus gravida. Proin elementum tempor ipsum eu cursus. Etiam imperdiet bibendum arcu. Vivamus sollicitudin efficitur vulputate. Curabitur eget leo sed sem ornare ultrices in in est. Nulla tempor congue libero, vel tincidunt sem sodales sed. Donec ligula orci, consequat eu sapien nec, placerat pellentesque ipsum. Nunc ut sapien elementum, porta erat et, convallis turpis. Nullam ac justo ornare ligula vestibulum pulvinar. Ut enim dui, suscipit auctor orci eget, finibus pharetra leo. In malesuada felis et elementum tempus. Nunc hendrerit lacus blandit mauris convallis laoreet. Suspendisse laoreet tempor lectus, eu lobortis arcu rhoncus et. Nam ut ullamcorper purus. Phasellus dignissim justo nec sem sodales, ut convallis lorem pretium. Etiam egestas orci nisi, quis pretium nibh laoreet ac. Fusce posuere posuere est. Suspendisse a sem facilisis, facilisis risus in, vestibulum justo. Aliquam erat volutpat. Vestibulum tincidunt enim in nisl varius, eu molestie nibh semper. Pellentesque eleifend a arcu non dictum. Morbi nisi lorem, pharetra maximus turpis a, gravida condimentum nisi. Proin ultrices blandit ante, sit amet laoreet ligula egestas finibus. Maecenas luctus, massa ut auctor accumsan, urna ex pulvinar lacus, ut iaculis lorem dui ac leo. Ut a arcu scelerisque, euismod sem sit amet, volutpat nisi. Ut ullamcorper bibendum nulla id cursus. Duis porta tempor urna. Donec vehicula sem et turpis scelerisque interdum. Proin nec tristique tellus. Mauris dapibus dolor sit amet suscipit feugiat. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nullam congue tristique quam, sit amet vehicula metus. Sed ante nisi, facilisis quis sapien vitae, ullamcorper gravida dui. Pellentesque nec arcu vel neque malesuada tempor. Morbi et sem eu nunc ultricies fringilla. Quisque ipsum lorem, efficitur non odio auctor, auctor porta est. Pellentesque tempor at enim a commodo. Vivamus eget gravida massa. Donec eget pulvinar ex. Nullam risus diam, malesuada et felis non, finibus mattis mi. Aliquam imperdiet aliquet velit, ac finibus libero eleifend vitae. Vivamus libero nunc, lobortis sed auctor at, tincidunt a ligula. Donec eleifend dignissim maximus. Duis luctus dolor vel viverra ultrices. Sed vestibulum eu felis non molestie. Morbi sit amet fringilla velit. Cras vel aliquet mi. Nam efficitur pretium tortor eu condimentum. Vivamus id tincidunt risus. Mauris libero lectus, aliquet id diam ac, egestas luctus velit. Nam pulvinar venenatis metus, eu malesuada lorem viverra id. Pellentesque porttitor quam in enim vehicula, et semper nisi ullamcorper. Curabitur vehicula turpis et blandit rhoncus. Proin placerat odio nisl, sed auctor quam volutpat vel. Fusce ultrices pulvinar imperdiet. Nulla sit amet leo sem. Integer ut nulla vel lorem semper pretium vitae in lectus. In neque augue, ornare eget vehicula non, efficitur non leo. Nam sed tellus eget erat posuere gravida varius eget diam. Sed interdum, lorem eget bibendum semper, velit turpis posuere felis, eget semper tellus ante ac ante. Fusce posuere ut lacus id elementum. Aenean ornare tempus orci non fringilla. Donec egestas velit quis varius elementum. Nam pellentesque tempus elit ut volutpat. Sed vehicula sem non nunc blandit convallis. Nunc nec libero ac mi feugiat maximus. Mauris tristique erat quis odio molestie, non aliquet ligula accumsan. Quisque lobortis ut elit sed iaculis. Maecenas dapibus congue tortor, ac sodales neque fermentum ullamcorper. Proin at libero diam. Suspendisse placerat ipsum quis scelerisque facilisis. Nullam quis tellus et eros scelerisque viverra. Nulla lacinia efficitur. Proin ultrices blandit ante, sit amet laoreet. All of this goes on a separate line causing the page break.

Figure 3.1l - With a keep property, the CSS formatter maintains the text in the same page

4 Experiment description and results

In the previous chapter we've laid out what needs to be considered important indicators for evaluating the capabilities of CSS and the CSS formatters. This chapter explores the way we used these indicators to evaluate the quality of the PDF files. Starting with PDFs of full books, book excerpts and crafted files all of them generated with multiple PDF formatters, we then apply different methodologies for evaluating the quality of the product, either manual or fully automatic checks.

4.1 Dataset

From our on the ground discussion with the publisher *Il Mulino* we gathered a group of PDF files, generated using XSL-FO. A formatter applies the editorial collection style, expressed in the form of typesetting instructions provided in a separate file, to the books. "An editorial collection in publishing, is a set of books published by the same publisher, with a set of similar characteristics"[32].

These PDFs can be grouped into two categories:

- public domain books, with a stylesheet from *Il Mulino* editorial collection.
- Book excerpts, previously used by the publisher to test the quality of the output from their tools and here used to compare them to the PDFs generated with CSS/HTML and the formatters in analysis.

Moreover we generated another group of test files to test additional behaviors not covered from the requirements and files of the publisher.

[32] "Una collana o collezione editoriale, in editoria, è una serie di testi pubblicati da una casa editrice con determinate caratteristiche comuni" - Ferretti, Gian Carlo, Iannuzzi Giulia. "Storie di uomini e libri : l'editoria letteraria italiana attraverso le sue collane" Minimum fax (2014).

4.1.1 Full books

The books used are:

- Cantico di Natale - Charles Dickens - 74 pages
- I Malavoglia - Giovanni Verga - 236 pages
- Il Manifesto - Karl Marx - 54 pages
- I promessi sposi - Alessandro Manzoni - 94 pages

The number of pages varies slightly depending on the style choices taken by the PDF formatter.

4.1.2 Test files from Il Mulino.

The publisher provided a group of tests, from which we selected four, organized in folders containing:

- An HTML file
- A CSS file
- An FO file
- The original output in PDF that we want to replicate

These tests are part of a larger test bed, which we used partially to cover the areas of interest for this document. The test bed will be used more extensively in the future as part of further research.

4.1.3 Crafted files

The last group of tests files, directly mapped to the indicators described in chapter 3.1, consists of manually generated HTML files, each one of them triggers a specific visualization problem in the generated PDF, allowing us to a) test whether in CSS exists a directive to address and correct the problem b) compare the result of multiple CSS formatters.

4.2 Tests and results

Starting with the input files described in Chapter 4.1, we executed two types of tests on them:

- Manual checks on short files
- Automated tests on full books, through a software written specifically for this task.

Table 4.2a summarizes the test title, whether it's an automatic test or a manual one, and the category of the input file distinguishing between the full books, book excerpts and manually crafted files.

Automated on full books	**Manual on book excerpts/crafted files**
Chapter titles at top of page (S1)	Title and body are kept on the same page (M1)
Chapter titles at odd pages (S2)	Long title is kept on the same page (M2)
Orphans not present (S3)	Title, subtitle and paragraph kept on the same page (M3)
Widows not present (S4)	Images and their captions are kept on the same page (M4)
Too much space at the end of the page (S5)	Text is kept in the same page (M5)
Too much space between text and notes (S6)	
Page never ends with a colon (C1)	
Title and body are kept on the same page (K1)	
Long title is kept on the same page (K2)	
Title, subtitle and paragraph are kept on the same page (K3)	
Images and their captions are kept on the same page (K4)	
Text is kept in the same page (K5)	

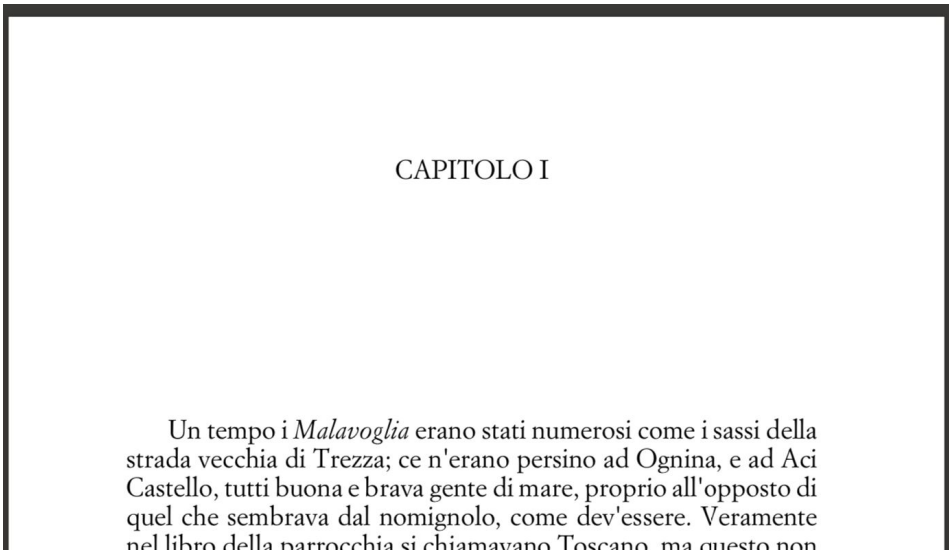
Table 4.2a - Tests description

4.2.1 Automated tests on full books.

As part of the search for an answer to whether CSS Paged Media is suited to print books, we produced a software to analyze a set of PDFs generated using various formatters, starting from four books available under public domain. We coded the automation to detect violations of some of the identifiers listed in Chapter 3 (see Table 4.2a, Automated) and parse all the given PDFs in search for errors that the formatter may have produced, either because of its own mistakes or because the CSS3 specification doesn't allow to produce that desired effect. We reverse engineered the provided PDF with the help of an FO file that was used to generate it. The CSS derived by this activity, could be then generally applied to all the books belonging to that specific editorial collection. Due to the peculiarity of each editorial collection and the less than perfect marshalling of the PDFs by the pdf parsing libraries, the code makes assumptions on how the document should look like. These assumptions are not relevant for the general case but the code can be improved and/or expanded to allow for the analysis of different real use cases. For details on the implementation methodology, refer to chapter 5.

Chapter titles at top of page (S1)

A chapter is a logical main division of a book, identified by a title that is distinguishable from the rest of the text by its position in the page and usually a different style. Given the different styles in use in different books and editorial collections, it's difficult to define a general programmatic rule to identify them. PDFMiner would parse the PDF in input and create a complex data structure, part of which is an array where every element is a paragraph. In this structure, a chapter title would appear isolated, i.e. not containing newlines, all uppercase and of size 10.7pt. This definition comes from the editorial collection style applied to all the full books under exam and the book excerpts (see Figure 4.2.1a).



CAPITOLO I

Un tempo i *Malavoglia* erano stati numerosi come i sassi della strada vecchia di Trezza; ce n'erano persino ad Ognina, e ad Aci Castello, tutti buona e brava gente di mare, proprio all'opposto di quel che sembrava dal nomignolo, come dev'essere. Veramente nel libro della parrocchia si chiamavano Toscano, ma questo non

Figure 4.2.1a - Example of a chapter title from one of the full books (*I Malavoglia*)

Chapter titles at odd pages (S2)

For ease of reading and better displaying, publishers use to print beginning of chapters in a page on the right. A chapter starting in a page on the left is considered an error. The CSS3 specification allows to define this, and formatters try to respect the setting with various techniques including leaving a blank page before the chapter starts. In the data structure created by PDFMiner when parsing the PDF, each page has a *pageid* which corresponds to the actual page number. Once a chapter title has been identified, is then trivial to check if it's found on an odd (right) or even (left) page.

Orphans not present (S3)

In typesetting, orphans are lines at the end of a paragraph, which are left dangling at the top of a column, separated from the rest of the paragraph. When an orphan is detected at the beginning of the page, this error is printed. PDFMiner parses the PDF creating, among other data structures, an array where each element is a string representing

paragraph. By counting the number of newlines, it's easy to understand the number of lines of the first/last paragraph and identify orphans and widows violations.

Widows not present (S4)

In typesetting, widows are lines at the beginning of a paragraph, which are left dangling at the bottom of a column, separated from the rest of the paragraph. When a widow is detected at the end of the page, this error is printed. Widows are identified by our software in the same way orphans are, by checking the number of lines of the first paragraph.

Too much space at the end of the page (S5)

For a better visual experience, page must be filled to the fullest. When too much whitespace is present at the bottom of a page, this error is displayed. PDFMiner provides the in-page coordinates of each element. By going through the array representing each page, our software identifies the coordinates of the lower corner of the last element in the page and calculates the empty space by difference with the page height.

Too much space between text and notes (S6)

For a better visual experience, page must be filled to the fullest. When too much whitespace is present between text and the notes on a page, this error is displayed. Similar to what was said for the previous test, the difference is calculated between the last element in the page and the coordinates of the top corner of the notes box.

Page never ends with a colon (C1)

A page must not end with a colon ':'. Since a colon "precedes an explanation or an enumeration, or list"12, it's visually unpleasant to see a page ending with it. This error message appears when a page ending with colon is identified.

Automated tests results

The software scanned the four full books given as input and produced the results in a CSV summary that we visualized in the following figures (4.2.1b-e). As it's visible, FOP, PrinceXML and PDFReactor all produce very limited number of errors that can then be addressed manually by the editor. As an example, think of the C1 test where the paragraph can be moved to the next page with a manually inserted page-break-before in the CSS, targeting an ad-hoc class naming in the HTML. Since a manual check from the editor is then needed, the process cannot be fully automated and is not to be considered optimal. It is worth noticing that while it's possible to configure parameters for the style and keep properties, the same is not true for the content properties (i.e. a page should not end with a column), so whether this appears in the text is just left to the case but correctly identified by the software when this happens.

We reviewed the software report and realized that the tool reported some false positives because of the way the PDF is parsed by PDF miner. In specific, no orphans or widows were present in the input files.

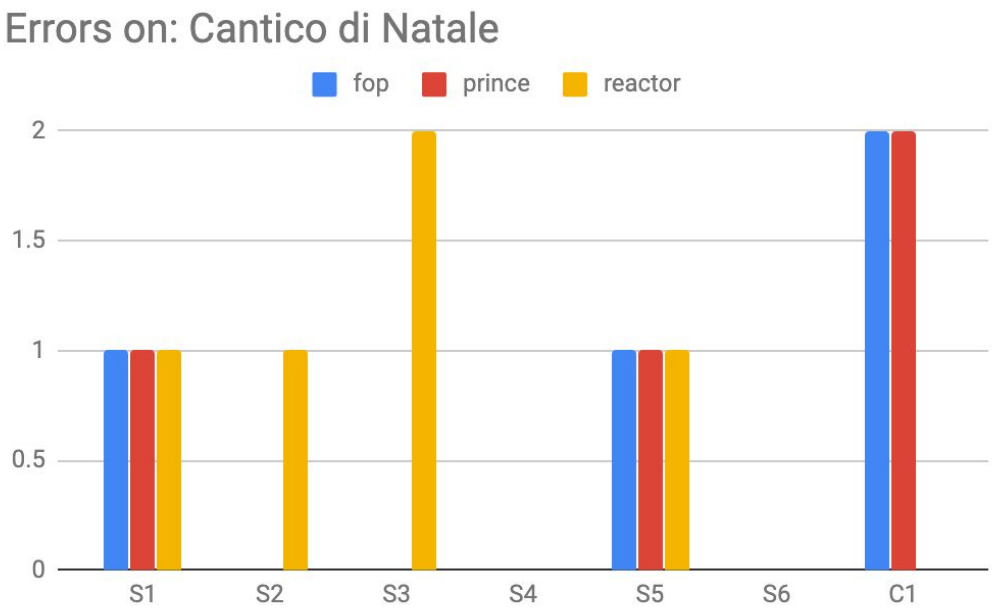


Figure 4.2.1b - Errors found in the book *Cantico di Natale* by formatter

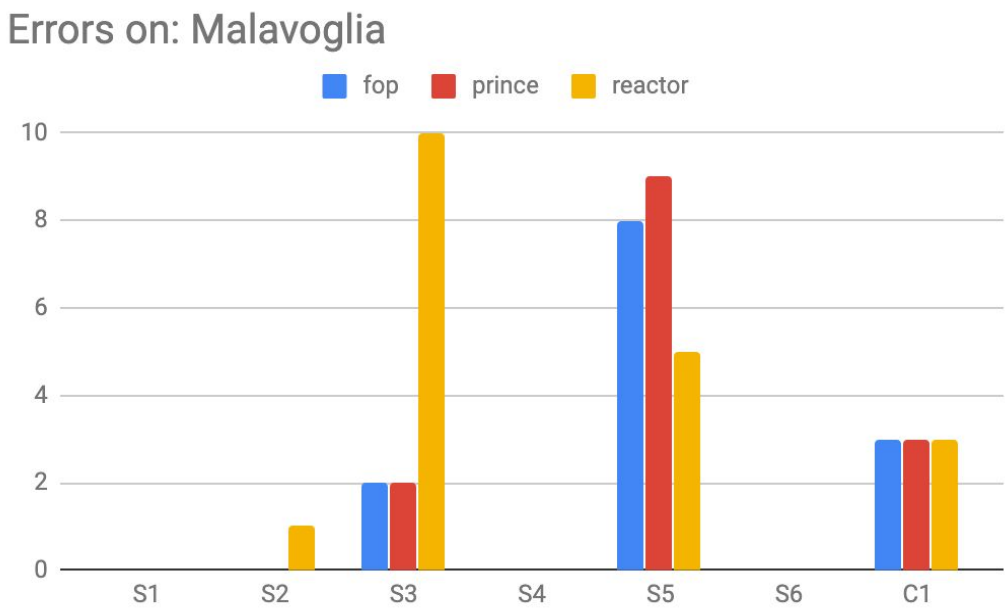


Figure 4.2.1c - Errors found in the book *I Malavoglia* by formatter

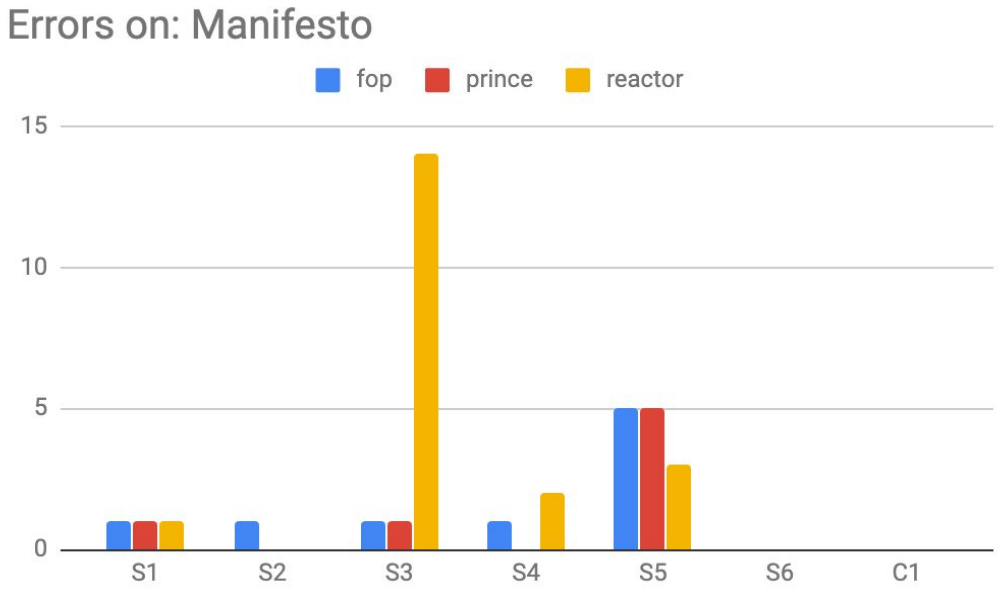


Figure 4.2.1d - Errors found in the book *Il Manifesto* by formatter

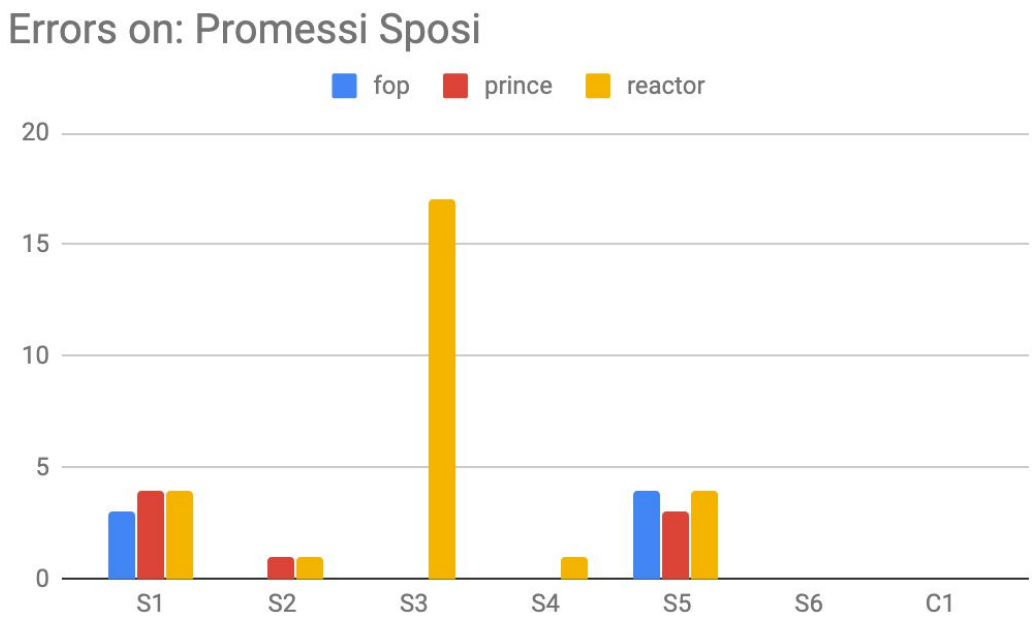


Figure 4.2.1d - Errors found in the book *I promessi sposi* by formatter

4.2.2 Tests from Il Mulino

Starting with the XSL-FO file and extracting from it the precise measures of distances, size, fonts etc., we were able to reconstruct a PDF that closely resembles the original to the point where the differences are minimal and negligible.

Blank page as last page (T1)

We aim at reproducing an XSL-FO generated PDF that contains numbered pages and ends with a blank page. The complexity lies in the lack of support in CSS for a blank page alone at the end which can be replicated by mangling with the HTML file, inserting an empty item and instructing a page-break-before: right; in the CSS.

Chapter levels (T2)

The source PDF file, generated with XSL-FO, contains several levels of chapter/paragraph titles. The relative spacing among the titles was produced in XSL-FO using conditional spacing, feature not supported by CSS. It is again possible to

reproduce the result in CSS but only by manually changing the class nomenclatures where the different spacing is needed. This implies additional work on the side of the publisher and the obvious lack of adaptability should the text structure change.

Images and captions (T3)

This book excerpt contains two types of images, differentiate by the spacing relative to the border. CSS fully supports this behavior.

Chapter title in a separate page (T4)

In this document we find a chapter title (bibliography) alone at the top of the page, followed by a blank page. Additional complexity is provided by a specific style for the list of books.

Results

In these tests we successfully replicated four book excerpt by “Il Mulino” to show how it’s possible to fine tune the HTML/CSS to remove even the minimal differences between the original file and the PDF generated with HTML. Being able to do so with the first two formatters (PrinceXML and PDFReactor) is proof enough of the capabilities of CSS3 and the work was not extended to additional formatters.

	PrinceXML	PDFReactor
Blank page as last page (T1)	✓	✓
Chapter levels (T2)	✓	✓
Images and captions (T3)	✓	✓
Chapter title on a separate page	✓	✓

Table 4.2.3 - Test titles and results

Note that PrinceXML ignores the directive on indentation for the footnote number and displays it in a different style at the very left of the page (See Figure 4.2.3a and 4.2.3b)

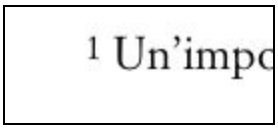


Figure 4.2.3a - Desired style and position

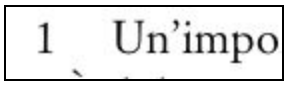


Figure 4.2.3b - PrinceXML output

4.2.3 Manual tests

Book excerpts, part of the dataset detailed in Chapter 4.1, differ from full books by their size and the purpose of checking specific FOP/CSS capabilities. An automated test of these files would have required a separate implementation for each of the indicators while a manual check, given the small number of pages and test bed, is fast and easy enough to not grant the effort for the automation. Refer to table 4.2.2a for a summary of the results of the tests for each formatter in analysis.

Title and body are kept on the same page (K1)

In this test, we check if and how a large text, by default being split in two different page, is treated by the formatters in the presence of a CSS directive that instructs them to avoid a page break.

Long title is kept on the same page (K2)

We forced the case of a multiline title at the end of the page, trying to trigger an edge case where a page break could have been inserted at the line break.

Title, subtitle and paragraph are kept on the same page (K3)

In the previous test, we validated that the keep property works on a single item like a title. In this one we check that it's possible to keep two items together (e.g. a title and the body of text) and that it's possible to expand the property to additional items in chain so that a defined group stays together in one page within certain limits.

Images and their captions are kept on the same page (K4)

Also in the case of an image and its caption, the default break can be overcome by applying the CSS rule for "break-after: avoid;"

Text is kept in the same page (K5)

When a text exceeds the size constraint of a page, the publisher might want to keep it together. CSS allows for this behavior with the "break-inside: avoid;" instruction. The tested formatters chose to sacrifice the constraint on the empty space at the bottom of the page to respect the keep rules.

Results

All formatters in analysis respond as expected in the presence of the keep directives even when stressed out with edge cases. It's worth noting that the test K2, which checks whether is possible to keep a long title in the same page, resulted in the

discovery that not only it's the default behavior of all formatters but also that is not possible to obtain the opposite result and force a page break inside a block.

	Title and body are kept on the same page (K1)	Long title is kept on the same page (K2)	Title, subtitle and paragraph are kept on the same page (K3)	Images and their captions are kept on the same page (K4)	Text is kept in the same page (K5)
AntennaHouse	✓	✓	✓	✓	✓
PrinceXML	✓	✓	✓	✓	✓ [33]
PDFReactor	✓	✓	✓	✓	✓
Weasyprint	✓	✓	✓	✓	✓

Table 4.2.2a - Key behavior support breakdown by PDF formatter

[33] While Prince honors the directive of break avoidance within the property and the text is kept together, to do so it breaks the "keep" rule between the title and the body, inserting a page break that leaves the title alone on a separate page.

5 Implementation details

In this chapter we explore the implementation of a software to check whether a given PDF is compliant with a set of aesthetic and formal requirements that we collected from precedent works on the subject and from the renowned publisher *Il Mulino*. This covers the automated tests performed on full books in the public domain, a subset of all tests performed. For the full list of tests, refer to Chapter 4.2.

5.1 Automatic PDF checker implementation

The code is written in Python3, language chosen for the wide availability of libraries to parse PDFs and for the familiarity of the author of this document. We'll explore more in detail the choice of the PDF parser library in the following chapters.

Since all our metrics and indicators apply to single pages and never to the PDF structure in its entirety, the code loops over each page, where they are passed to separate functions, one per indicator and the relative metric is incremented if the criteria is met.

At the end we print a report, on screen with visual structure and on file in CSV format for further data analysis.

The initial function in the code works just as a wrapper and container for the boilerplate that loads the PDF in memory and parses it with pdfminer. In its central function (process_pdf), the code goes through page by page and for each one of them executes the following checks:

Chapter titles at top of page (S1)

A chapter is a logical main division of a book, identified by a title that is distinguishable from the rest of the text by its position in the page and usually a different style. Given the different styles in use in different books and editorial collections, it's difficult to define

a general programmatic rule to identify them. In the case in study a chapter is defined as a standalone alphanumeric string, whose characters are all uppercase and font size 10.7pt.

```
if chapter_not_at_page_start(layout):
    logging.warning("Page {}: Contains chapter not at
start".format(layout.pageid))
    chapters_not_at_page_start += 1
```

```
def chapter_not_at_page_start(layout):
    first = True
    for element in layout:
        if isinstance(element, LTTextBoxHorizontal):
            for obj in element:
                # skip the first element.
                # If it's a chapter, then it's correctly placed.
                if first and is_chapter(obj.get_text()):
                    continue
                if not first and is_chapter(obj.get_text()):
                    return True

                first = False
    return False
```

Chapter titles at odd pages (S2)

For ease of reading and better displaying, publishers use to print beginning of chapters in a page on the right. A chapter starting in a page on the left is considered an error. The CSS3 specification allows to define this, and formatters try to respect the setting with various techniques including leaving a blank page before the chapter starts.

```
if layout.pageid % 2 == 0 and contains_chapter(layout):
    logging.warning("Page {}: Contains chapter not on the
right".format(layout.pageid))
    chapters_not_on_right += 1
```

Orphans not present (S3)

In typesetting, orphans are lines at the end of a paragraph, which are left dangling at the top of a column, separated from the rest of the paragraph. When an orphan is detected at the beginning of the page, an error is printed.

```
if is_orphan(layout, orphans):
    logging.warning("Page {}: Orphan
identified".format(layout.pageid))
    orphans_num += 1
```

Widows (S4)

In typesetting, widows are lines at the beginning of a paragraph, which are left dangling at the bottom of a column, separated from the rest of the paragraph. When a widow is detected at the end of the page, this error is printed.

```
if is_widow(layout, last_page_ends_with_dot, widows):
    logging.warning("Page {}: Widow
identified".format(layout.pageid))
    widows_num += 1
```

Too much space at the end of the page (S5)

For a better visual experience, page must be filled to the fullest. When too much whitespace is present at the bottom of a page, this error is displayed.

```
if too_much_space(layout, limit=MAX_SPACE):
    logging.warning("Page {}: Too much space at the end of the
page".format(layout.pageid))
    pages_too_short += 1
```

Too much space between text and notes (S6)

For a better visual experience, page must be filled to the fullest. When too much whitespace is present between text and the notes on a page, this error is displayed.

```
if too_much_space_between_text_and_note(layout):
    logging.warning("Page {}: Too much space between text and
note".format(layout.pageid))
            space_notes_text +=1
```

Page never ends with a colon (C1)

A page must not end with a colon ‘:’. Since a colon "precedes an explanation or an enumeration, or list”12, it’s visually unpleasant to see a page ending with it. This error message appears when a page ending with colon is identified.

```
if is_last_char_colon(text):
    logging.warning("Page {}: Ends with colon".format(layout.pageid))
    last_char_colons += 1
```

5.2 Code limitations

The code used for the automatic PDF parser is not flawless, as software usually is. The limitations are present not only in the software itself but also on its dependencies, like the pdf parser library, the formatters and the requirements, vague by definition.

5.2.1 Software implementation

The code produced is well organized in short functions that are easily testable. Unit and integration tests are though missing and would greatly improve the speed at which it’s possible today to integrate additional tests and PDF formatters. Moreover some of the specification, deeply tight with the requirements from *Il Mulino*, like the limit number of

widows and orphans or the space limit at the bottom of the page, are hardcoded and should be moved to a configuration management system.

5.2.2 PDF parser library

An additional research is needed on the PDF parser libraries. PDFMiner was selected among the python libraries for the better than average results but still the resulting data structure is overly complicated to navigate and requires extensive usage of debuggers to overcome the lack of documentation.

5.2.3 Requirements

Requirements coming from publishers are very different in nature and in most of the cases specific to each publisher if not often to each separate editorial collection. This makes it difficult to cover all use cases and calls for a separate research on the most common cases to expand the test suite covered in this work.

6 Conclusions

In this document we analyzed the possibility to use CSS and HTML as a solid successor of XSL-FO with three test suites:

- A combination of HTML/CSS to generate and check individual behaviors in PDFs
- book excerpts from the publisher “Il Mulino”, generated originally with XSL-FO and that we managed to replicate accurately with HTML and CSS.
- complete books to which we applied an editorial collection stylesheet derived from an XSL-FO source.

For each HTML/CSS source, we used all the formatters in analysis to generate PDFs, which gave us an overview of the market status in terms of support of CSS Paged Media.

While book excerpts and crafted short PDFs were easily checkable manually, for the larger books we recurred to a software we developed to automatically scan the pages and check for errors.

Most of the commercial PDF formatters’ output is up to expectations, respecting the CSS Paged Media specifications and adding proprietary options to ease the work of printing. Among them, PDFReactor is the one that made our work the easiest, producing a PDF that is best parsable by PDFMiner and thus perfectly analyzable programmatically. Opposite to that, AntennaHouse’s watermark, while perfectly legit, made it impossible to work in conjunction with our software.

The open source space is full of options for PDF mangling, mostly aimed at PDF editing and text extraction. It is our opinion that lots of work is still needed to reach the level of the commercial offering, to the point that most of the software we tried was not worth including in the present work of research.

This document is not definitive and more can be done to expand it. Given enough time we hope that the gap between the open source offering and the commercial one will be reduced enough to consider them comparable.

Overall, we were able to demonstrate how CSS Paged Media supports all of the requirements and output capabilities of XSL-FO either directly or, in a few cases, with manual intervention on the HTML/CSS code. Specifically, XSL-FO is designed with the concept of page sequences, lacking in CSS which instead works on individual pages. This different paradigm resulted in the inability to insert blank pages at predefined positions, if not in relation to a preceding page.

Bibliography

1. Adler, S. (2000). *Extensible Stylesheet Language* (XSL)-Version 1.0.
2. Berglund, A. (2006). *Extensible Stylesheet Language (XSL) Version 1.1.* Retrieved from https://www.w3.org/TR/xsl11/ on June 2019.
3. Bos, B. (2011). *All CSS specifications.* Retrieved from http://www.w3.org/Style/CSS/specs on June 2019.
4. Carter, R. (1993) *A widowed line, highlighted in yellow*.
5. Ciancarini, P., Di Iorio, A., Furini, L., & Vitali, F. (2012). High-quality pagination for publishing. *Software: Practice and Experience*, *42*(6), 733-751.
6. Day, B., Meggs, P. (1993) *Typographic Design: Form and Communication*. John Wiley & Sons, 263
7. Etemad E. (2018). *Selectors Level 4*, Retrieved from https://www.w3.org/TR/selectors-4/ on June 2019.
8. Ferretti, G., Iannuzzi G. (2014). *Storie di uomini e libri : l'editoria letteraria italiana attraverso le sue collane.* Minimum fax
9. Grant, M., Etemad, E. J., & Sapin, S. (2013). *CSS Paged Media Module Level 3.*
10. Jung, A. (2019). *Print-CSS rocks*. https://print-css.rocks Retrieved on June 2019
11. Justus, P. E. (1972). There is more to typesetting than setting type. *IEEE Transactions on Professional Communication*, (1), 13-16.
12. Knuth, D. E., & Plass, M. F. (1981). Breaking paragraphs into lines. *Software: Practice and Experience*, *11*(11), 1119-1184.
13. Knuth, D. E. (1986). *Computers & typesetting*. Reading, MA: Addison-Wesley.
14. Mittelbach, F. (2019). A general framework for globally optimized pagination. Computational Intelligence, 35(2), 242-284.
15. Pawson, D. (2002). *XSL-FO: making XML look good in print.* "O'Reilly Media, Inc.".

16. Shinyama, Y. (2004). *PDFMiner*. Retrieved from https://euske.github.io/pdfminer/ on June 2019
17. Sneep, M. (2005). A short comparison of various typesetting engines. Retrieved from http://web.archive.org/web/20070203144903/http://www.nat.vu.nl/~sneep/ars/type/comparison.pdf on June 2019.
18. University of Chicago Press. (2010). *The Chicago manual of style*. University of Chicago Press.

Formatters

- *AntennaHouse Formatter v6*. https://www.antennahouse.com/formatter/
- *PrinceXML* - https://www.princexml.com/
- *Apache FOP*. https://xmlgraphics.apache.org/fop/
- *PDFReactor* - https://www.pdfreactor.com/
- *RenderX tools index*. http://www.renderx.com/tools/index.html
- *Weasyprint*. https://weasyprint.org/
- *WKHtmlToPDF*. https://github.com/wkhtmltopdf/wkhtmltopdf/issues/2066

CSS Specifications

- *Break-after.* https://developer.mozilla.org/en-US/docs/Web/CSS/break-after
- *Break-before*. https://developer.mozilla.org/en-US/docs/Web/CSS/break-before
- *Page-break-after.* https://developer.mozilla.org/en-US/docs/Web/CSS/page-break-after
- *Page-break-before*. https://developer.mozilla.org/en-US/docs/Web/CSS/page-break-before