Single Source Publishing

A investigation of what Single Source Publishing is and how this 'holy grail' can be achieved.

Published: Aug 16, 2021
Author: Adam Hyde
Illustrations: Henrik van Leeuwen
PDF Design: Harshna Haswani
Rendered from HTML using Paged.js

Note: This is a combination of 6 articles published in a series in 2021.

The Problem

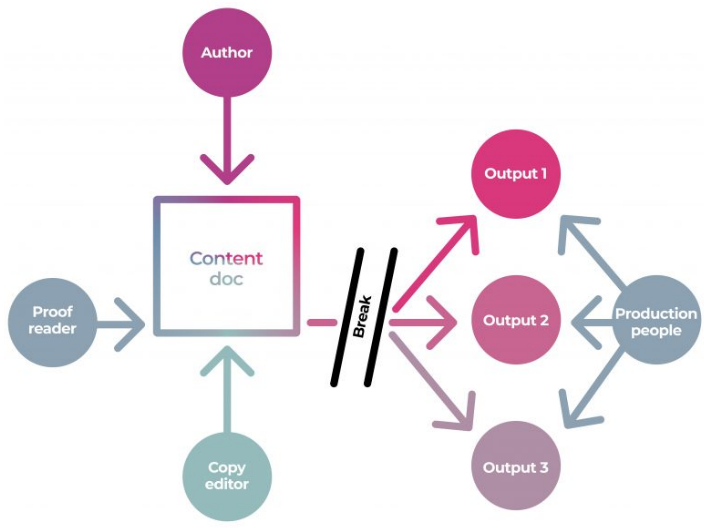
Publishing has long suffered from broken workflows that slow down the time to publish and unnecessarily inflate the cost to publish. Publishing of all kinds suffers from these inefficiencies. Some of the problems are general, others are very idiosyncratic and particular to perhaps just one publisher.

However, no matter what kind of publishing you do, there is a good chance you suffer from a disconnect between content creation processes and production processes.

In many publishing environments authors, copy editors, proofers etc create and improve content primarily in one tool (usually Microsoft Word). The content creation process feeds into the production process where production staff create a multitude of other formats – HTML, PDF (for print and screen), possibly XML and ebook formats etc.

To convert to these file formats, the content has to either be programmatically converted to various target formats via software (eg Pandoc or bespoke software), or manually converted by people using software (eg InDesign). If manually executed Publishers either sling the content 'over the wall' to a publishing services vendor or contractor, or they employ internal staff. The folks doing these conversions belong to the general category of 'production people' – usually programmers, designers, or 'format wranglers'.

Here is the problem. Workflows that look similar to the above, which is most publishing workflows, separate the content creation from the content production. This disconnect separates 'the who' (who does the work), 'the how' (what tools are used), and importantly 'the what' – what files are worked on. The people, the tools, and the working files all change as the content jumps from content creation to content production.



To understand why this is a problem, we just have to consider this one very simple scenario: what if there need to be changes to the content after it has entered the production stage?

Generally, if content needs to be changed while in the 'production stage' it requires several steps

1. the content people, using their own tools, communicate the changes required
2. the production people, using their own tools, make the changes for each output format
3. the content staff check the changes.

All this also requires managing because there is a lot of necessary back and forth which in turn creates a lot of expensive overhead for making even the simplest of changes. Additionally, jumping the gap from creation to production introduces errors.

Take a simple book example – if an error is discovered by the author (which is common) after the content has gone to the designer, then the author must write the comments somewhere (email/MS Word/annotated PDF), and send them to the designer. The designer must then interpret the results, which can in itself cause errors as designers are seldom content domain experts, and apply the corrections in (for example) InDesign. The designer then sends a PDF to the author to check..and so on...

Or a simple Journal example – the production staff have discovered from proofs that figures are in the wrong place. This information must be communicated to (for example) the publishing services provider (via email, MS Word, annotated PDF etc). The vendor interprets the information, makes the changes in their (various) tools, and sends back to the publisher to check... and so on...

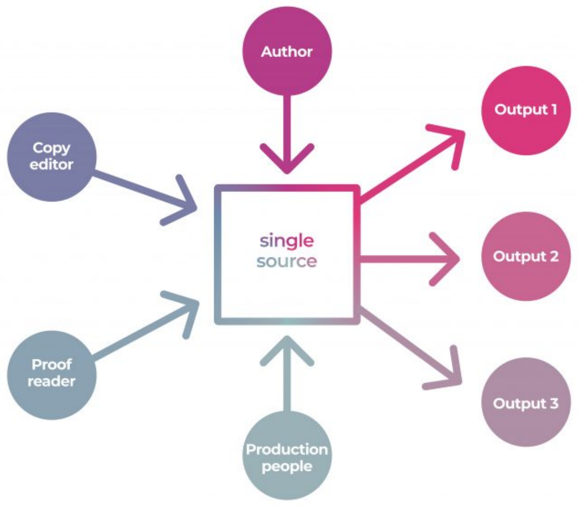
In each of the above examples, larger publishers also have staff to manage the communication of changes, track the changes, check the changes etc. It is quite an ordeal that costs time and money. If publishers don't do this well, the consequence is that more errors are introduced.

This is the problem 'single sourcing' is meant to solve. Single sourcing is a general approach to publishing workflows that is intended to avoid disconnecting the content creation and production processes – saving time and money, and reducing errors.

What is Single Source Publishing?

What exactly is single sourcing and how does it avoid disconnecting content creation and production?

Single sourcing isn't a specific solution, it is a general idea that must be intentionally designed into a publisher's workflow. Single sourcing changes how people work and often requires a different tooling. The secret really, if we zoom out to a high-level abstraction of the problem, is to work out how the content creation and production folks can work in a shared 'environment' where they all work on the same files, the same source files – hence the term 'single source'.



In a single source environment if a change needs to be made while the content is in production the content people can make the change themselves. Less time and communication required, less to-and-fro, less management overhead, and fewer errors.

There are many ways of achieving single sourcing. Any book publisher could solve this by, for example, asking their authors to write their books in InDesign, saving the files to a shared server somewhere. Authors, copy editors, and designers (etc) could all collaborate in the same tool, changing the same files, and all changes could smoothly flow into the final output at the push of a button. Success!

There is a good reason why you haven't heard of anyone doing this despite the fact that this is a *technically* elegant single source solution. No author is going to head off into the woods and spend 6 months in a cabin to conjure up their masterpiece – in InDesign. Not ever. InDesign is a production tool, it is not designed for authors.

The above example may seem frivolous, but it gets down to the core of the problem. Single sourcing, to be effective, must be done with tools that not only share the same source for content creation and production but also respect the different tool-cultures of content creators and production people.

Amazingly, this last point is often not taken into consideration. Unfortunately, too often, the content creators are never asked about the kinds of tools they like to work with. Their tool-culture is ignored. This is actually a problem with how tooling is designed and built, and worthy of one or more articles in itself.

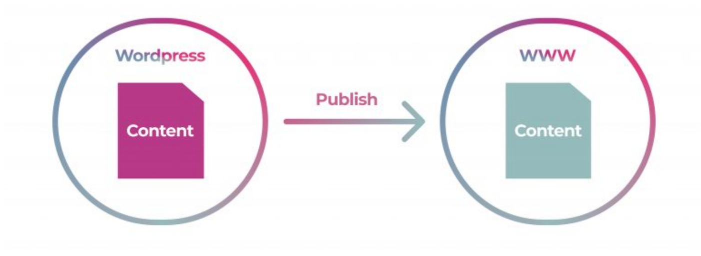
A Solution

Let's look at a simple tool used for single source publishing on the web – WordPress – which will help us understand some of the more complex problems Publishers face.

WordPress is a single source publishing system. To produce a post, the author does the following:

1. writes the post within WordPress, using the basic web editor supplied
2. presses 'Update' and the content is pushed through to the web

That is it. The important thing to understand here is that content is authored in one place, and at publish time, that content is transformed into the target output – in this case a blog post shared on the web.



Let's add a little complexity as it is hard to really understand the value of single source publishing from such a simple example. We will add more people and more outputs.

Single Source and Multiple Actors

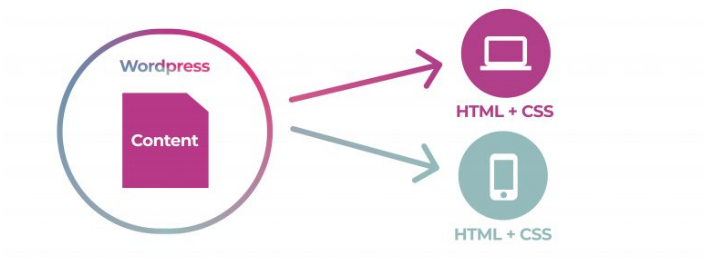
Before publishing, other actors – copy editors, illustrators etc – might also work on the content. These folks will also work within WordPress to edit the same post. All these people can collaborate on the same source for the post. When it is time to publish, the process is still as simple as pressing publish. There are no other versions of the content that need to be merged, checked, managed etc. Single source has, even in this basic example, saved us a lot of hassle.

Furthermore, all the stakeholders here are using familiar tools. The WordPress editor is a pretty easy editor to use if you are creating content for the web.

Single Source and Multiple Outputs

Now let's look at adding multiple outputs. What if we wish to display the same WordPress blog post to look good on both mobile and desktop devices. In this case, CSS (the set of rules used by designers that describe how webpages should look) has the ability to identify whether the content is being displayed on a small or large screen. CSS changes how the content is displayed accordingly.

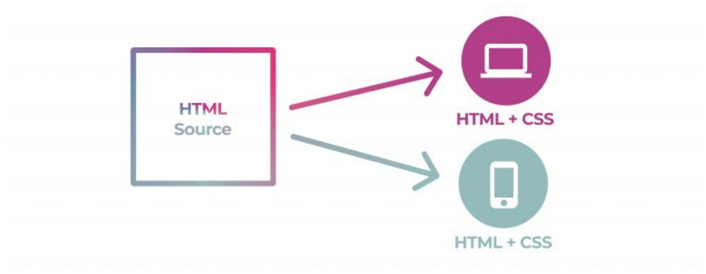
As if by magic, we have effectively solved the problem of displaying the same content for two different types of outputs – desktop and mobile displays.



So what's the problem?

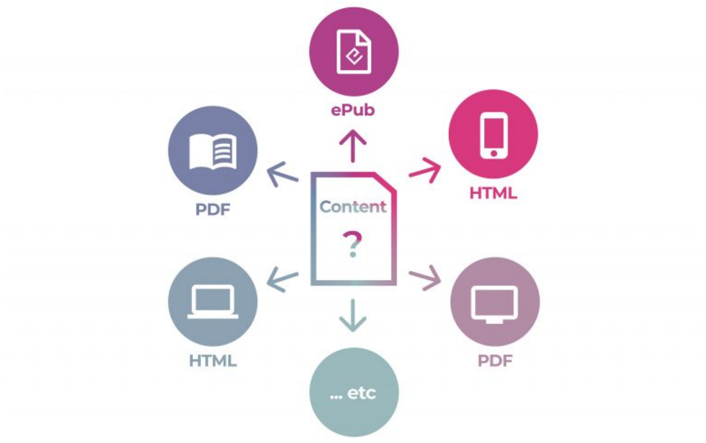
If you were building WordPress for the first time, and you needed to cater for the above outputs (HTML + CSS for 2 display sizes), what file format would you choose to store your source?

No prizes for choosing HTML. To transform from HTML to HTML & HTML is cheap because there is (in simple terms) no transformation needed. This is exactly what the smart folks at WordPress have done.



The content you are editing in WP is stored as HTML. Exactly that same source is directly transferred to the web at publish time.

The trouble starts when you require significantly different types of outputs. What if, for example, you required not just HTML but also EPUB, Screen PDF, PDF for Print, XML etc. In cases where you require multiple output types, which is most publishing scenarios, you must choose a source file format that can undergo significant transformation into all of the output formats you require.



So, how do you choose the right source file format for your publishing system?

When designing a publishing system you face two basic questions, in this order:

1. what output formats do I need?
2. what source file would best lend itself to transformation into the outputs?

After these two questions there are a lot of secondary questions (requirements) that will come to bear on the problem, but let's keep it simple for now.

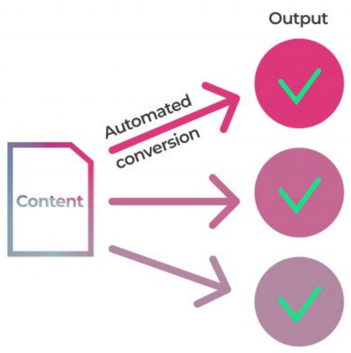
Unfortunately, the process for designing systems too often takes the following format, in this order:

1. what source file format is currently in fashion? (hint: it has been XML for the last 20+ years)
2. how do I build a system around that format?

Unfortunately, this latter thought process has driven the publishing industry into building expensive and inefficient systems and largely prevented the industry from moving towards more elegant workflows.

Is Automation the Answer?

Let's look a little closer at the automation of file format conversion. Obviously it would be awesome if we could push a button in our imaginary publishing system and all the outputs would roll out automagically – fully formed, perfect and beautiful.



Is such automation possible? Let’s have a closer look at the types of conversion involved:

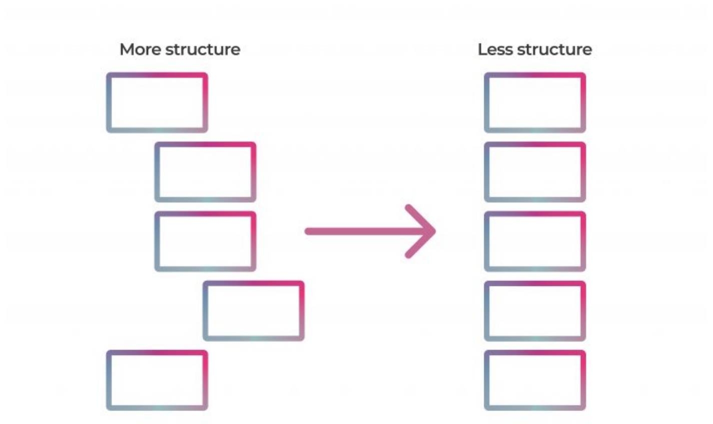
1. Structural Conversion
2. Typesetting

Structural Conversion

There are two types (roughly) of structural conversion – upconversion, and downconversion.

Downconversion

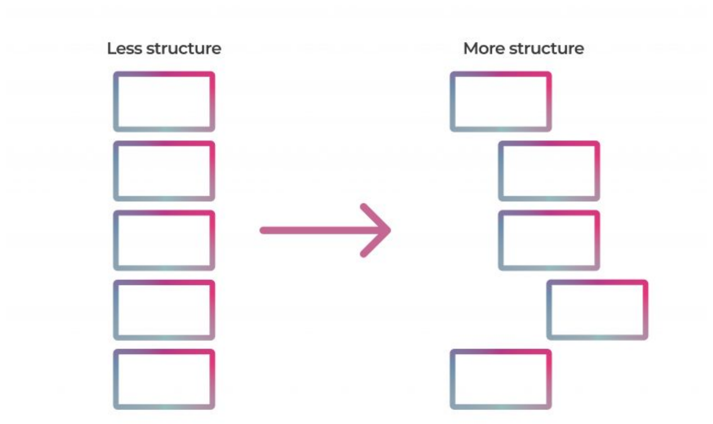
To create a simpler structure (eg plain text) from any source format, the transformation mostly means removing information (structure).



Removing structure is easy (usually), we just throw stuff away.

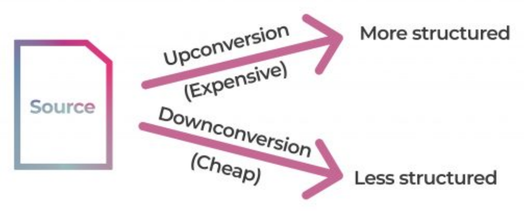
Upconversion

To get from our source file to a more complex structure, we need to add information (structure).



Adding structure is doable but not as easy.

A simple diagram to help us understand up and down conversion would be as follows:



Down-conversion is generally an easier target for automation. Up-conversion is more likely to require human intervention.

Typesetting

Another category of conversion is typesetting. Typesetting means we require a change in the look and feel of the output (design).

There are two approaches to typesetting – ‘automatic typesetting’ (which is done by a machine) and manual typesetting (done by a human).

Automatic Typesetting

Automatic typesetting is when a designer sets up a bunch of rules (templates) and then ‘a machine’ applies those rules to the content.

Automatic typesetting is possible to achieve in some Publishing contexts but the more complex your design requirements, the harder it is to achieve.

Manual Typesetting

When a designer uses a design tool such as InDesign to produce the required design, this is manual typesetting.

Again, the more sophisticated the design, the more likely it is to require human intervention to produce the desired result.

Where does that leave us?

If we can achieve all of our conversions – structural conversions and typesetting – automatically, then we are in a winning situation. We can start with a source file and then press a button and all of our outputs will be generated automagically. This is single source publishing achieved through automation.

Automatic conversion is most likely to be achievable where we have either:

1. very simple conversions
2. moderate expectations from our conversions.

While often true in ‘web publishing’, these two conditions are seldom true for publishers. Publishers, generally speaking, have high expectations for all their conversions and their outputs are often complex.

What does this mean? In most publishing contexts, this level of automation is not achievable – we will require manual processing, and manual processing inevitably leads to the broken workflows we were trying to avoid. If, for example, we want to introduce a manual design tool such as InDesign, then we are changing the people, tools, and source as discussed above and we are deciding against a single source workflow.

If we are to achieve single source publishing, but we can't do it by automation alone, then we need another strategy.

To achieve all the efficiencies of the single source workflow that we have discussed before, we need to work out how humans (content producers, designers, production people) and machines can utilize different tools, respecting their tool-cultures, while working on the *same source* to create all the desired outputs. How is this possible?

For the Good of The System

> "[XML].. is seen as something that must be endured by content authors for the greater good of the enterprise."
> Peter Meyer, 2005

It seems (from above) that our dream of perfect automated output is dead, so how do we ensure all the folks involved in preparing content to publish – authors, editors, copy editors, designers, format wranglers etc – can work on the same source?

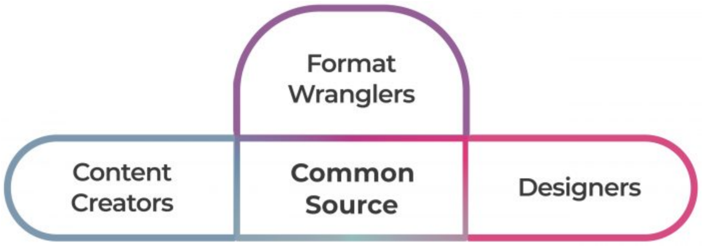
Let's look, at a somewhat simplified level, at the operations each of the main categories of stakeholders have to perform.

Content – ability to change the content

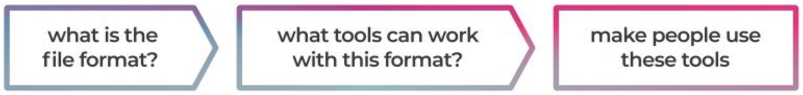
Design – ability to change the look and feel of outputs

Format – ability to change the structure of outputs

The challenge for systems designers is to consider tooling that can help each group of stakeholders work efficiently while they share the same canonical source.

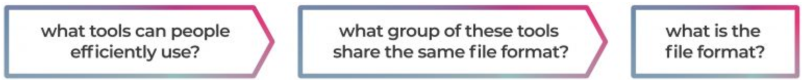


To solve this problem, the publishing sector's 'system thinking' has largely been, to date, driven by what experts believe is the best file format. This 'file-format-first thinking' has looked like this:


what is the file format?
what tools can work with this format?
make people use these tools

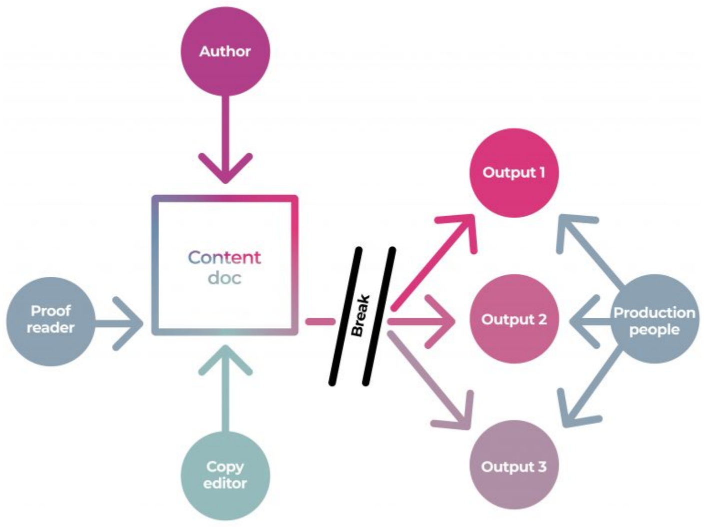
This thinking has lead us to where we are today, and today we have broken, disjointed workflows.

To bring about the efficiencies we are after, we need to think in this direction:


what tools can people efficiently use?
what group of these tools share the same file format?
what is the file format?

What if the file format was the last decision you made when constructing ecosystems of publishing tools, rather than the first thing you decided?

Are we more interested in making decisions based on how we can help people work more efficiently, or making decisions based on source file formats? When designing systems the publishing sector has, for the past 25 years, appeared to have answered "source file formats" to this question. This is largely why the sector has been driven away from single source publishing systems and is mired in the kind of broken, slow, expensive processes we discussed in earlier and we illustrated as follows:



When we started our discussion of single source publishing systems in this article, we talked a lot about source file formats. However, it appears that deciding on the actual source file format to be used is (literally) the last question we must ask. The questions we need to ask are as follows, in this order:

1. what tools can people efficiently use?
2. what group of these tools share the same file format?
3. what is the file format?

We need to start looking at *ecosystems of tools* that people can use efficiently to work together on the same source, and then adopt that source format, whatever it may be..... this might be a lengthy exercise as there are a lot of tools and formats out there, so I'll make it easy on you.

Workflow-First Systems

In the above section, we concluded that it is a very good idea to build systems around ecologies of tools that share the same source file format, rather than deciding on the source file format before workflow needs are considered. Rather than a 'file-format-first' approach, we should move towards a 'workflow-first' approach.

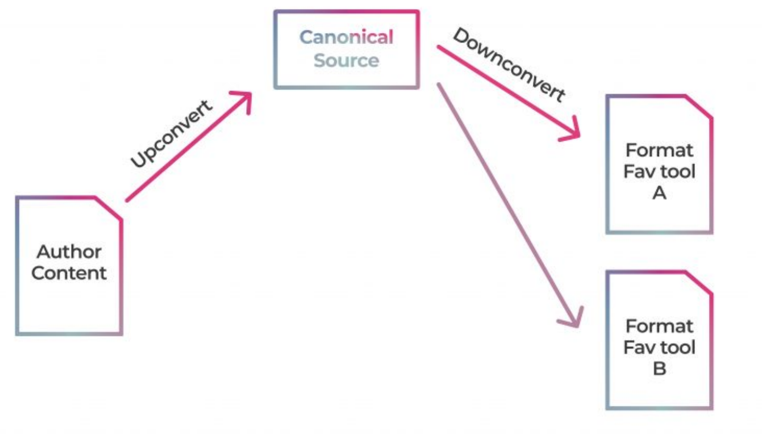
So we need to first consider, at a somewhat simplified level, the operations each of the main categories of stakeholders in a single source publishing system have to perform:

Content – ability to change the content

Design – ability to change the look and feel of outputs

Format – ability to change the structure of outputs.

Setting aside the search for tools for a minute, it might be reasonable to ask – is this approach even possible? Don't each of these operations require something quite different from a file format? So don't we require different formats to service each phase? It seems, in the history of publishing so far, systems designers have appeared to believe this to be true. This is why many publishing systems upconvert to the 'highest resolution' format possible (generally some form of xml) as early as possible, and then downconvert to specific formats for consumption by tools such as InDesign.



But it is possible for a single format to contain enough information to feed into each of these operations. Let's look again at each category of operations:

Content

First, the content folks must be able to edit a relatively easy-to-understand document. Authors, etc, in most cases prefer a document which contains only what is known as 'display semantics' as they don't generally work with tools that enforce structure, rather they write from top to bottom with a structure that is meaningful to them via the headings, indents etc displayed in the actual document.

Authors of this kind, which is most authors, 'just like to write'. They don't care too much for having to maintain anything beyond the text, other than how the document looks to the eye.

So any format we choose must be able to support a suite of tools that these kinds of authors can use. Generally speaking these tools are known as 'word processors'.

Design

Designers must be able to take the same content and apply design to the content within the constraints of the output format. For example, designers must make the document look good in paginated PDF for printing (eg books), or EPUB, or the web etc.

Any suite of tools we choose must enable designers to change placement, color etc of all the elements in the content as well as controlling the same for format-specific features (eg running headers, page numbers etc) for each output type while using the same source file. Generally, to date, the typical tools for designers have been what is sometimes referred to as 'pixel pushing' tools – tools where you can point and click to target elements and change their look and feel. However, in recent years (well, for a while now) there have been 'rules based' design tools. One such rule-based approach is CSS – the set of rules web designers use to determine the display format of webpages.

Format

Format wranglers want to be able to output file formats required for archiving, transmission, and storage. JATS (Journal Article Tagging Suite) is one such format, it is a variant of XML. Books have BITS (Book Interchange Tag Suite) which is also a XML variant.

Format wranglers need the source file to contain enough information so the content can be translated (restructured) to the new form. Format wranglers are a kind of technician—someone that is expert in encapsulating data in logical structures. The tools for these folks have generally been specialist pseudo-scripting tools such as XSL (Extensible Stylesheet Language) which is part of the family of XML tools. However, there are other approaches – JSON, for example, is often used as a way to represent data, especially for transferring or storing data for web applications. With JSON, transformation is managed by rules encased in custom JavaScript code. But in essence, good format wranglers like to write rules that can transform many files, they don't like manually transforming each file.

For a format to meet the needs of each of these use cases, there are two factors that must come into play:

1. any alterations to the content by content, design, or format people, MUST affect only one source – the single source
2. the source file format must contain enough information to service each of the tools used by these folks

That seems to make sense… but there is one additional issue that is very important and which may not be so obvious. Any of the tools used can of course augment the source file with information for the job at hand while not requiring that information to actually be part of the source file.

A good example is with transformation. The transformation folks can write a set of rules that can map the source file onto another structure (such as JATS). To do this they need two things:

1. enough information in the source file to enable the transformation. For example, to map A->B the source file must first tell us what A is .
2. a set of rules that manage the execution of the transformation

The interesting thing is that the rules ([2] above) for the transformation do not have to be contained within our source file. These rules can exist entirely independently of the source file.

In this way the format wranglers can do a lot of things no one else cares about, without overloading the source file with all kinds of requirements. This helps us keep the amount of information the actual source file needs to contain to a minimum.

This is, if you had not realized this already, exactly the opposite to the way publishing systems designers have approached this process to date.

So, let's say this makes sense so far. But what of content creation and design?

For editing/content creation we need a format that doesn't break if it is 'badly structured'. You may not know it, but if there is one thing authors are very good at, it is badly structuring documents. So, we need a format that doesn't care if the author structures things oddly. We can clean that aspect up later.

This brings about an interesting quality of the file format. It must be capable of being *progressively structured*. That is, we can start with a 'badly structured' document and the source file format won't break. We can then improve the structure over time and the file format will also be happy to do this.

This is also, not the way it has been managed to date in publishing systems.

Now to design. Design requirements are very similar to the requirements for format wrangling. We need the source file format to hold just enough information so that we can design elements with our design tools, but any further logic can be contained within separate rules and not contained within the source file. Once again, that allows us to keep the requirements of the source file to a minimum.

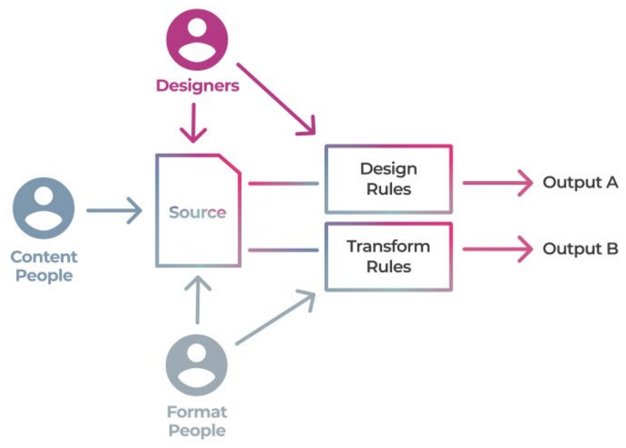
Ecosystem Features

So, we have two sets of qualities when choosing our ecosystem — one set of features for the source file format, and one for the types of tools we choose. For the file format we need this:

1. contains just enough information for each of the operations (create, transform, design)
2. can be progressively structured.

For the tools we need this feature:

1. the design and transformation are managed by logic external to the source file format.



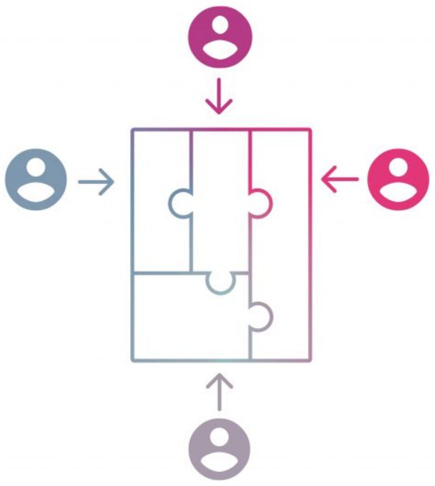
Where does that leave us? As it happens, it leaves us with the ecology of tools that surround one of the most popular formats of our age – HTML.

Concurrency

In publishing there are two types of concurrency we care about:

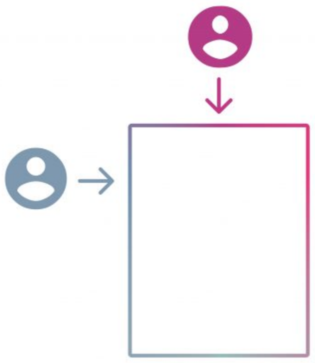
1. workflow concurrency – the ability to perform multiple tasks upon the work, by different people, at the same time
2. realtime editing – the ability for multiple people to edit a document at the same time and see each other's changes in realtime (like Google Docs, or Plasmic / Figma etc)

These two overlap but they are distinct. Here is a brief example to illustrate the difference. In a book production workflow, we may break the work (book) up into multiple documents – one for each chapter or even per paragraph (sometimes referred to as batching). Just by doing this we can have more than one person working on the book at the same time (each tackling a different chapter for example).

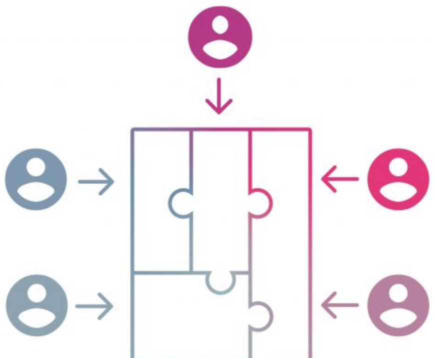


This is workflow concurrency.

Now, if we want two people to edit the same chapter at the same time this is realtime editing.



Of course we can blend these two models:



Now, realtime editing, as a type of concurrency, is a choice you have depending on the type of technology you choose for editing (there are also concurrent design apps like Plasmic and Figma). If you choose to use Microsoft Word for content creation, for example, you cannot achieve realtime editing. If you choose other tools (eg Google Docs, or ProseMirror and Firebase / ProseMirror and yjs) then you *can* achieve simultaneous realtime editing.

But it is good to remember that if the technology does offer realtime editing then there is nothing forcing you to use it. Whether you use it or not is dependent on how you want people to work. Previously in this series of articles about single source publishing workflows, I suggested we do not design publishing platforms starting from technical features (such as choosing the source file format first), but rather we take a workflow-first approach. It is the same with realtime editing – first work out your ideal workflow, and if indeed you actually want realtime editing, then add that to your list of system requirements. The technical consequences are a secondary problem.

As for workflow concurrency – we have the same set of questions. Do we want multiple people to perform different types of tasks on the work at the same time? Do we want, for example, the designer to be working on the layout as the copy editor does their work? Or an illustrator to be designing illustrations while an author is commenting on the illustration at the same time? These workflow concurrency questions must be answered when choosing or designing a publishing system, and once again it comes down to your answer to the crucial question – "what is my ideal workflow". Other decisions follow from there.

Given all this, it is a good idea to first zoom out and question what value concurrency, of any kind, can offer a publishing workflow. We can then answer whether we actually want concurrency.

Concurrency vs Sequential Workflows

Publishing workflows these days are mostly sequential. A sequential workflow requires operations to be executed one after the other. Many publishers today, for example, are emailing (or uploading and downloading) bunches of documents for each other, usually in the MS Word format, to work on. An author emails files to a acquisition manager, who emails them to the production crew, who back and forth with authors and copy editors etc to get all the files processed (or "done_done_final_done_really-final-done_v2" as the case may be) and then the same files are sent to a designer/vendor etc

This is very sequential and we know its downside because that is how things work today. We know how much time and money these workflows cost (much more than necessary).

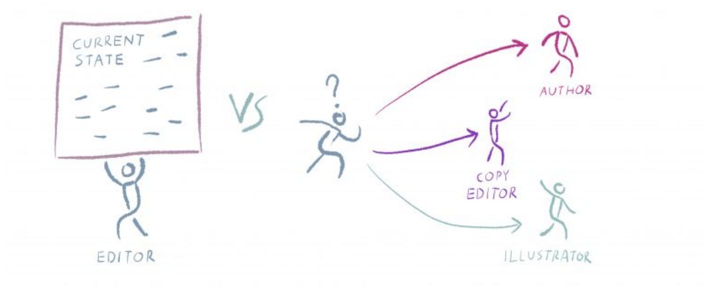
The question is whether concurrency is better? To answer this question we have to understand the essential characteristics of concurrency, and how they can help publishers.

There are two essential characteristics of concurrency:

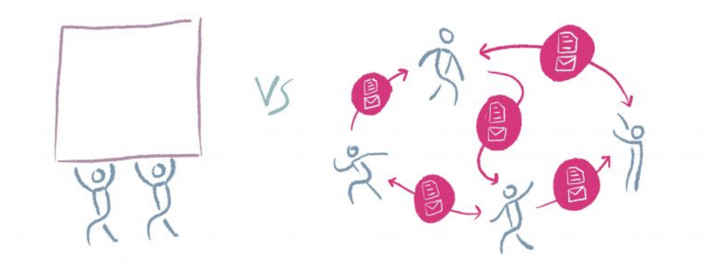
1. the opportunity to see the current state of the work at any moment
2. the opportunity to act on the current work at any moment.

There is a lot to gain from these two simple features. From a very high level there are 4 general gains on offer that are derived from [1] and [2] above:

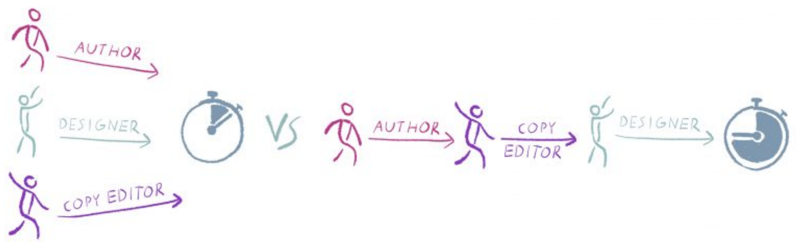
1. **understanding state** – an immediate understanding of the state of the work at any moment



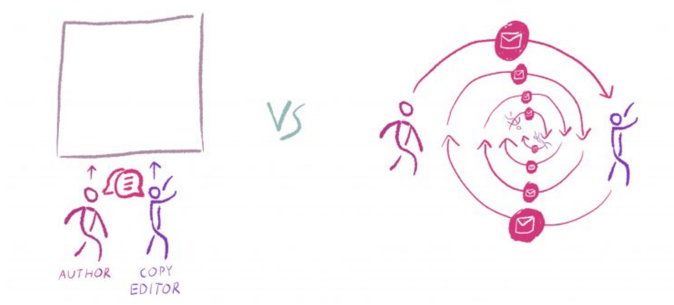
2. **reducing handling time** – reducing the to and fro of passing documents back and forth



3. **parallel processes** – various team members can do what they need to do in parallel with each other



4. **realtime problem solving** – issues can be resolved in realtime.



These 4 opportunities provided by concurrency can have, even if moderately implemented, a huge impact as Ken Brooks has attested:

> “By implementing collaborative editing, WYSIWYG rendering, and push-button distribution to all formats (courses, eBooks and print files) we discovered it can result in a 50% reduction in cost and a 50% reduction in cycle.”
> —Ken Brooks, president, Treadwell Media Group.

But concurrency also allows for completely new ways of working, as Barbara Rühling from Book Sprints points out:

> “Book Sprints would not be able to produce books in 5 days if authors, illustrators, copy editors, designers etc were not all able to work concurrently. ”
> — Barbara Rühling, CEO, Book Sprints

How much concurrency benefits your workflow depends on which parts of the cycle you decide to make concurrent, and just how far you are prepared to go. How far you take it is up to you.

Here are some examples of where concurrency could benefit a common publishing workflow:

- Art logs and image research starting as soon as FPO (‘for placement only’) images are placed
- Alt text being created as images are being added

- Indexing could start as the text is completed (inserting tags) before the proof stage
- Proofing being finished on one part of the book before the rest is finished
- Concurrent page rendering enables author to correct or fit content as they write it
- Testing the content with live customers as the rest of the content is being finished (O'Reilly does this, and it's a way to do A/B testing for learning objects and assessment items in education. It's a way to improve product-market fit).
- Multiple authors working on the same text at the same time
- Designers working on design of the actual content as a work is being created
- Copy editors working on content immediately as it is completed by an author
- Illustrators placing images in a work as it is being authored
- Production editors understanding exactly what has and hasn't been done at any time
- Developmental editors and authors working on the same text at the same time

SSP and Concurrency

So, what about SSP and concurrency?

Remember these two features of concurrency listed above:

1. the opportunity to see the current state of the work at any moment
2. the opportunity to act on the current work at any moment.

These two features require the source to be shared between all those in the team. To have concurrency, on a system or document level, we need to share the source … sound familiar? Sharing the source is the same core design requirement of a single source publishing system. SSP offers more opportunity by design for concurrency than does a 'multiple master' publishing system.

SSP can offer publishing enormous gains through the introduction of concurrency into the workflow. Concurrency can help optimise how you work, or it can radically change how you work. How much you take advantage of SSP and concurrency is completely up to you. The advantage of SSP is that you have a choice, as Andrew Savikas points out:

> "Concurrency is so powerful. It means it's possible to (for example) have an author, copy editor, indexer, and designer all making changes in different parts of the manuscript at the same time. Of course, it does not always (or even often) make sense for each step of the workflow to be done concurrently, but an SSP model at least makes it possible. "
> — Andrew Savikas, Chief Strategy Officer at getAbstract

Thanks to Wendell Piez, Ken Brooks, Andrew Savikas and John Maxwell for feedback. Thanks also to Henrik van Leeuwen for the images and Raewyn Whyte for copy editing.