The Swiss Cheese Problem, or, Chesterton Redeemed:
Being a response to, and enlargement upon, his long-awaited opus, “The Neglect of Cheese in European Literature”, through a comparison of the formal qualities of caseinetic foodstuffs to problems in Library Economy.

Jacob Nadal
Executive Director
ReCAP
On Cheese...
On volumetric inefficiencies in cheese...
On Switzerland and its national dish...

Rub the pot with garlic
Heat wine and a touch of lemon juice
Mix with your starch, preferably corn or potato
Add cheese slowly, stirring and melting
Finish with a dash of *kirsche* and a little nutmeg
On the arrangement of items on shelves...
The “Swiss Cheese Problem” of library shelving

• Removing individual volumes from high-density storage creates random gaps
  • Gaps can only be filled by a same-size item
  • Thus, manual selection and placement of items is required
• And the cost per item in labor is high enough that it is less expensive to simply build additional storage space
ReCAP Quantified: 11.3 Million Items
(and growing at a rate of some 50,000 each month)

- 2012 OCLC Study [http://recap.princeton.edu]
  - 442,422 duplicates + 46,575 triplicates from 4,437,546 monographs
    - 9.9% duplication + 0.1% triplication = 10%
  - 471,593 duplicates + 48,880 triplicates from 4,851,089 titles
    - 9.7% duplication + 0.1% triplication = 9.8%
- So let’s assume 10% item-level duplication
  - From 11,300,060 items: 1,130,006 dups
Skepticism

• Can this be true?
• Is it really the case that it’s less expensive to build space for 1,130,006 books than to backfill existing space?
• Are we just going to give up!?
R. I. P.
Bold Spirit of Librarianship

Dawn of Recorded History -- 2014
Tray (a) is reshelved, the duplicate is removed from tray (b), and tray (c) is introduced.
Newly arrived books are added into the partly filled trays (b) and (c), and then accessioned
Items do not have a standalone workflow (Items are accessioned in trays, the actual unit of work)

But, are these item costs that we can implement?

This kind of analysis abstracts out some key limits (materials handling, in this case) and accidentally assume a zero-cost, zero-friction shift from one unit of work to another.

This is an accidental “best case” analysis. If we started work on this basis, we’d go over time and budget. Seem familiar?

<table>
<thead>
<tr>
<th>Cost /Item</th>
<th>$ 1.32</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/o sizing</td>
<td>$ 1.23</td>
</tr>
<tr>
<td>partial comparison</td>
<td>$ 0.91</td>
</tr>
<tr>
<td>w/o comparison</td>
<td>$ 0.49</td>
</tr>
</tbody>
</table>

If large batches of same-size materials are coordinated, sizing time may not matter

If comparison is only when the item fails a condition check (50%)

If no comparison is required

But, are these item costs that we can implement?
Trays do have an existing workflow

**Standard Rate:** *These are from actual operations at ReCAP*

- Retrieve: 65/hour
- Reshelve: 65/hour
- Sorting (on return): 100/hr
- Sizing: 1.25 batches/hr
  - 1 batch = 1 shelf full of trays
  - Trays/shelf varies w/ tray size
- Accession, Verify, TTS: 50 trays / hour

<table>
<thead>
<tr>
<th>Duplicate Resolution Rate</th>
<th>Minutes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieve Tray (a)</td>
<td>60</td>
</tr>
<tr>
<td>Retrieve Tray (b)</td>
<td>60</td>
</tr>
<tr>
<td>Sort Items</td>
<td>39</td>
</tr>
<tr>
<td>Reshelve (a)</td>
<td>60</td>
</tr>
<tr>
<td>Trays to Reaccesion</td>
<td>60</td>
</tr>
<tr>
<td>Size Trays (3 – 4 batches)</td>
<td>180</td>
</tr>
<tr>
<td>Accession</td>
<td>60</td>
</tr>
<tr>
<td>Verify</td>
<td>68</td>
</tr>
<tr>
<td>Tray to Shelf</td>
<td>60</td>
</tr>
</tbody>
</table>

**Hours for 65 trays:** 10 hrs

**Mins/Item:** 9.29

**Cost/Item:**
- $3.15 w/o sizing
- $1.01 partial comparison
- $1.25 w/o comparison
- $1.06

After deriving an item cost from the actual workflow, the costs are **much** higher when sizing is factored in.

Sizing starts with a large set of effectively random materials that are sorted into like-size batches. That takes time a lot of time.

On a tray-basis, if we eliminate sorting or sizing, we get closer to the item-basis projection.

Sorting and sizing work across trays, so once again, we have two units of work in our projection.

So when we pull duplicates, can we blunt the impact of sizing by pulling same-size materials?
On the nature of things singular and plural...

- 1,130,006 dups (est.)
- 843,709 trays
  - 1.34 dups / tray
  - 5 dups / 4 trays
- 80,035 shelves
  - 14.12 dups/shelf
  - 113 dups / 8 shelves

- These rates suggests several possibilities
  - Must be trays with multiple instances
  - May be large clusters of duplicates

- "Hot spot" mapping
  - less travel during retrieval
- Whole (serials) trays that can be removed
  - instant open space with no downstream work
- Large shelf areas reopened
  - faster to refill
- Large same-size batches of duplicates
  - reduces work of sizing, our most expensive step
On the likelihood of a positive return on effort

Per Volume Costs

• For 442,422 monographs:
  • At $1.25 \rightarrow $552,893
  • At $3.15 \rightarrow $1,392,300

Even with cost overruns up to 300% on the low end, and 40% on the high end, this is a worthwhile effort. These are suitable projected conditions for a useful pilot project.

Space reclamation v Construction

• $2,000,000 to build shelves
  • Marginal benefit of $600,000 to $1,450,000
• In addition to cost savings:
  • Verify shelving accuracy (better than 99.99% confidence with a margin of error below 0.002%)
  • Pushes back date of next construction (frees two aisles in the existing facility)
Conclusions and Outright Speculation

Conclusion:
• Deduplication of fixed location storage can show a positive return on effort

Speculation:
• We have the right process (plan, pilot, project)
• We need to be more rigorous in planning
  • Formalized and repeated at different scales
• Planning is tedious, can be protracted, but still cheap
  • Professional hourly rates look expensive, but labor and unit costs accrue en masse ➔ scale matters once again
  • Labor and unit costs are hard to contain once a project starts