



Thinking Lucene ▼ Think Lucid.

---

# Starting a Search Application

Marc Krellenstein  
CTO  
Lucid Imagination

---



## Abstract

If you think you need a search application, there are some useful first steps to take: validating that full-text search is the right technology; producing sets of ideal results you'd like to return for a range of queries; considering the value of supplementing a basic search results list with document clustering; and producing more specific requirements and investigating technology options.

There are many ways that people come to the conclusion that they need a search application, and a variety of ways in which they can then proceed. Here are some of the steps that users often consider (or maybe should consider) before they're ready to start building an application.

## Table of Contents

Do You Need Full-text Search?.....	1
Identifying Ideal Results.....	2
Clustering Results .....	5
Producing Requirements and Choosing a Technology.....	7
Resources .....	9

## Do You Need Full-text Search?

Full-text search is useful when you want to find something in a large amount of unstructured text—documents, articles, product descriptions, case studies, informal notes, web sites, discussion boards, wikis, e-mail messages, job descriptions, resumes, patents, legal decisions, etc.

A “large” amount is usually much more than you can conveniently scan or read from start to end to find what you’re looking for, and it could potentially be as much as millions or billions of items. “Unstructured text” refers to the fact that the text consists of ordinary sentences, sentence fragments, or words of some kind rather than numeric values, structured alphanumeric values (e.g., account IDs, product codes) or non-textual content

“What makes full-text search useful is that some of the data you are looking through is unstructured.”

such as pictures, video, and audio. The unstructured text you’re searching will usually exist together with structured “fields” or related pieces of information—the price of a product, the date of an article. But what makes full-text search useful is that some of the data you are looking through is unstructured.

Full-text search is good at a variety of information requests that can be hard to satisfy with other technologies. These include:

- Finding the most relevant information about a specific topic, or an answer to a particular question,
- Locating a specific document or content item, and
- Exploring information in a general area, or even browsing the collection of documents or other content as a whole (this is often supported by clustering; see below).

Full-text search is useful even if you have a way of navigating to a particular item of interest. Early on, Yahoo! pioneered the idea of a directory of websites, allowing users to

select what they want by drilling down in a hierarchy from a general category (“geographic regions”) to successively more specific ones (“downtown Seattle”). However, most users seem to prefer search even when a browsing option is available. This is partly a matter of personal style, but also reflects the fact that an accurate search system may allow a user to find information faster than they could by browsing, (e.g., a search on “downtown Seattle” may get you what you want faster than browsing by country, then state, then city, then city area). Of course, users may sometimes want to browse – especially when they are just exploring different content areas – and at other times want to search. It’s readily possible to combine these in one system, as is discussed below in the section on clustering search results.

Full-text search is less effective when you always know exactly what answer to provide. Some full-text search systems (such as open source Solr) let you “rig” specific documents as answers to certain matching queries. But the greater value of such systems is in the way they can quickly find the mostly likely answers to an ad hoc request that may have no predefined right answer. If all or nearly all the information requests can be answered with specific documents or content, users might be better served by an FAQ or some other system that directly links specific requests to specific answers.

A full-text search system may also not be the best choice when unstructured text search is secondary to more structured searches. In those cases a relational database may be a better technology choice, especially since most such databases today provide some form of full-text search for their unstructured text fields. (See the article on choosing a search engine or a DBMS, in the Resources section at the end of this paper.)

## Identifying Ideal Results

If you’ve concluded that a full-text search system is what you need, you’re going to have to understand the data you’re searching and the users who will be doing the searching. You will then need to design the search application to maximize “findability” of different kinds of information for those users on that data. (Grant Ingersoll has a good article on optimizing findability in your search application; see the Resources section at the end of this paper.)

However, system designers sometimes jump too quickly into the middle of the requirements or the design process without being clear about just what they would most like the search application to do. To figure that out, you might create a basic “paper

prototype” of some search scenarios: Take a pencil and some blank paper and, for a number of possible user queries you expect, sketch the content items you would ideally like the system to return to the screen. You should put these ideal results in the actual order in which you want users to see them. Include any specific content for each item you’d like returned on the search results screen—a title, summary and/or any other particular attribute of the item. Don’t try to specify the graphical specifics of the user interface—layout, color, formatting—but just the specific content you would ideally like returned.

This is often not an easy task. It can require scouring your data by whatever means you have available (existing search systems or databases, browsing individual documents, etc.) to determine what the possible results would be and carefully considering the user’s request to figure which are the best results.

“Because this is often difficult, many tend to skip this step ... [but] you can’t really design for effective relevancy ranking if you don’t have a goal”

Because this is often difficult many application developers tend to skip this step, preferring just to build a system and improve it over time to get better results. People may also skip this step because they assume that ideal results are unattainable. While that is often true, it’s not always the case. If you don’t at least start with what you’d ideally like to find, you may have already made an unnecessary compromise, and you can’t really design for effective relevancy ranking if you don’t have a goal.

Determining what you most want to get in light of the user’s query will also give you important clues about what technology you need and how to build the system. Matching results to a query is usually a combination of finding the results most similar to what the user is asking while also favoring results which may be inherently “better” by some measure in a particular context, for example, more authoritative, more recent, longer (or shorter).

To determine the best results and give you some insight into the system you’ll need to build, you may want to consider some or all of the following:

- What exactly is the content you want to return (and where is it located and how will you be able to gather and process it for search)?
- How varied is the data? Are there many different kinds of information? Are you able to intelligently order the results into one list or do multiple lists, corresponding to different data sources, make more sense? Is the content of varying length or size and, if so, how does that influence what results are best?
- What are the direct connections between the content and the query, such as matching keywords or phrases? What transformations of the queries or content (case normalization, synonyms, translation, etc.) are you assuming to help find those ideal results? Is the structure of the content matter—titles, summaries, layout, etc.—important for matching purposes? Is deeper analysis of the query and/or content—recognizing complex entities such as specific people, locations or events, or relationships among them—implicit in what you think are the best results?
- What, if any, are the indirect connections between the query and the results that bear on the quality of the results? Does the source or authority of the result affect its value as an ideal result? Do you as system developer want a say in which results are best? Is third-party information available and useful in determining the results (e.g., that other people have linked to or cited that result; or, that people who have performed a similar query, and then spent time reading specific results)? Are there attributes of the user—level of expertise, interests, job description, etc.—that determine what the ideal results are?
- How important is it that the information be recent? Do you want results sorted by date, or is date just one factor in what makes a result good? Or is it not important at all? If recency is important, how different would the ideal results be a minute, hour, day, or week later?
- Are there many acceptable answers or only one or a small number? If there are many, is it important to return all of them? That would suggest recall is important—that it's important to get most or all relevant results, even if that means getting some irrelevant ones (as is usually the case if you want to see most or all good results).
- Do you only want to see very relevant results or are you also interested in somewhat relevant ones (and don't mind some irrelevant ones)? If you want to see only relevant results that would suggest that precision is especially important—returning relevant results even it means missing some other relevant ones in order

to minimize seeing irrelevant ones. (It also is usually the case you'll be missing some good ones if you don't want to see too many bad ones).

- Will certain users only have permission to see certain results?
- Are there so many results that typical users will need additional tools to manage or view them, such as clustering results into categories? See below for more on clustering.

## Clustering Results

There are many ways in which a basic search application can be supplemented beyond simply providing an ordered list of results. One common extension is the clustering of search results into categories of related results. This is useful when even the best relevancy ranked list of documents will not suffice to give users what they want. This may be because the query is too general to identify what the user is really interested in or because the user is not interested in something specific but is just exploring an area of interest. In other cases you may be interested in discovering certain common themes in the results, or that a particular portion of the content has relevance.

In all these cases, supplementing a best effort relevancy ranking by also presenting groups or clusters of documents organized around a specific topic or common characteristic can give a user greater visibility and ease of navigation over the full set of results. In these cases choosing from a set of categories organized by a common subject, date, price range, or other attribute will often be faster for getting the user to what they want than walking a long results list from beginning to end.

“Clustering results by a common subject, date, price range, or other attribute will often be faster for getting the user to what they want than walking a long results list from beginning to end.”

The situation is analogous to what happens when a user begins requesting information in some real-life situations, such as at a store or library. To get what you want often requires a dialogue with someone knowledgeable, and it's just the need for simulating such dialogue that was one early motivation for document clustering. If someone goes into a store and asks for a television or into a library and asks for information on baseball, producing the single best television or baseball book is really a wild guess. The store clerk might first ask "how big a television?" and the librarian might ask "What about baseball? I have general information, statistics, team information, books about how to play baseball, the World Series, the World Baseball Classic...." The user might respond to the librarian with "I want information about teams." And the librarian's response might be "the Red Sox?" (because you're in Boston), or "the Yankees?" (because people in Boston hate them), etc. The user might also learn about an aspect of the game—the existence of a World Baseball Classic—of which they were unaware.

A system that clusters documents responds like a (somewhat inarticulate) librarian. Instead of a real dialogue in response to a query on baseball, the system responds with the best documents it can but also blurts out (displays) some possible clusters: history, teams, statistics, etc. Then the user might select a cluster—teams—and the system responds both with more specific documents (the query has been effectively narrowed from baseball to baseball teams) and by clustering the now more specific results. Clusters then might be Red Sox, Yankees, etc.

The user can keep selecting categories (and/or entering new queries against the now smaller set of documents) until the request is specific enough that the documents are what the user is looking for. Of course, the user may not have had a very specific question in mind, in which case the successive clusters and documents offer a way to explore the subject that the original results list doesn't really provide (short of reading the whole thing). In some cases of exploration, users may want to start not with a search at all but by browsing the entire collection, starting with the top-level categories and drilling down to more specific areas of interest (a situation where something like Yahoo's directory – which is a top level list of document categories—can be particularly valuable).

Note that the goal is usually still to get to the best or some particular documents of interest. Where this can be more readily achieved, clusters may be of less value. Even when clusters are of value they are sometimes ignored by users, since the current dominant search paradigm pushes the users toward review of the results list. For this reason, clusters should not take the place of doing the best possible relevancy ranking, but they can

supplement that, especially when it's difficult to determine (even for a human) what makes up a best ranking or when discovering new aspects of something is important.

Some studies have shown that about 10 percent of queries result in selection of a cluster when clusters are available. However, regular users tend to use clusters at least occasionally, and the perceived value of clusters—even when they are not widely used—may contribute to the perceived value and actual use of the system.

(You may want to see Yonik Seeley's article for information on faceted search in Solr; faceting is a particular form of document clustering. See "Resources" at the end of this paper.)

## Producing Requirements and Choosing a Technology

After determining a search application is needed and considering what the system should ideally return, the next steps are usually to produce more detailed requirements and choose a technology. These are sometimes intertwined. Certain core requirements may dictate certain technologies, but a particular technology may then have capabilities that suggest other application goals.

Any of the above characteristics of ideal results, including the possible value of clustering, will drive your more specific requirements and may affect what technology is best for your application. Different technologies provide different advantages for integrating or searching varied data, for transforming data or queries, for exploiting third-party information, for handling dates or frequent updates, for favoring recall or precision, for providing security or for grouping results.

**“One useful way both to drive requirements and evaluate a technology is to build a proof of concept or prototype of the intended application.”**

All of these are points to include in your review of technologies, in an RFP, and/or in discussion with technology providers. Things like cost, support, performance, scalability,

reliability, likelihood of further improvements and customizability may also be important factors, as they often are for any technology decision.

Loosely speaking, there are some technologies—such as Lucene/Solr—that are relatively generic and handle most or all of these capabilities well or very well. Such technologies are usually strongest in the “basics”—strong and flexible ranking of documents, excellent query performance and the ability to scale for large systems—and may be functionally best for most applications or if some more specific need can be handled within them. Technologies with a more specific focus may be more useful for more specialized needs, though they might still do well enough on the basics for more general solutions if there is some particular (nontechnical) reason for choosing them.

At one time, users looking to build search applications often struggled with the question of “make vs. buy,” that is, should they build their own search system from scratch or start with technology from an existing vendor. “Make” decisions were mostly motivated by the perceived lack of good existing systems or by strong needs for control or customizability to produce exactly the right application.

Today there is usually little reason to consider a pure “make-from-scratch” alternative. There are a number of good quality technologies, and the existence of a high quality open source alternative—Lucene/Solr—generally satisfies any strong need for control or for customizability, since the user has and controls the underlying source code but without the cost or risk of writing it from scratch.

Sometimes, one useful way both to drive requirements and evaluate a technology is to build a proof of concept or prototype of the intended application. This will usually be done on a limited set of data, with limited user interface or packaging, and/or with limited functionality, perhaps focusing on areas where there is uncertainty about whether a needed capability—query or content transformation, accuracy of results, speed or scalability—can be provided. A basic prototype may also serve as a first iteration in an “agile” development model in which the final product is built less by an upfront, top down requirements and design process and more by successive iterations that include increasingly more functionality, content, and/or polish, as seems appropriate at each step.

## Resources

- “Search Engine versus DBMS,” by Marc Krellenstein. Discusses the choice between a search engine and a relational database for your application:  
<http://www.lucidimagination.com/Community/Hear-from-the-Experts/Articles/Search-Engine-versus-DBMS>
- “Optimizing Findability in Lucene and Solr,” by Grant Ingersoll. Discusses designing your application to maximize the ability of users to find what they’re looking for:  
<http://www.lucidimagination.com/Community/Hear-from-the-Experts/Articles/Optimizing-Findability-Lucene-and-Solr>
- “Faceted search with Solr,” by Yonik Seeley. Discusses the faceting (a form of clustering) available in Solr:  
<http://www.lucidimagination.com/Community/Hear-from-the-Experts/Articles/Faceted-Search-Solr>  
<http://www.lucidimagination.com/Community/Hear-from-the-Experts/Articles/Faceted-Search-Solr>