Requirements:

A search engine that allows the user to run queries against a collection of documents. It describes both an abbreviated requirements analysis and the resulting requirements specification.

**Normal case behavior:**

Suppose the user starts a session with the search engine. The first question that comes up concerns whether the engine already has a collection of documents that it remembers from the last time it ran. Let’s assume that the customer is interested only in new searches. Therefore, **the first thing the user must do is identify some documents of interest. Let’s assume that this is done by presenting a URL of a site containing documents**; the engine will run searches against all of those documents. Furthermore, the customer is interested in multisite searches; therefore, the user can present additional URLs of document-containing sites, and the engine will increase its collecition as a result. The customer indicates that the collection can be enlarged at any time, not just at the start of a session, and that there is no interest in removing documents from the collection.

The customer indicates that **a user should be able to search the collection for a document with a particular title**. However**, the main purpose of the engine is to run queries against the collection,** which means we have to decide what a query is. In consultation with the customer, we determine that a query begins by the user presenting a single word, which we will refer to as a keyword. The customer indicates that many words are uninteresting (e.g., “and” and “the”) and will not be used as keywords. The customer expects the search engine to know what the uninteresting words are without any user intervention; thus, it must have access to some storage, such as a file, that lists the uninteresting words.

The system responds to a query by presenting information about what documents contain the keyword. This information is ordered by how many times the keyword occurs in the documents. The system does not present the actual documents, but rather provides information so that the user can examine the matching documents further if desired.

However, the ability to query using a single keyword is quite limited, and **the customer also requests the ability to “refine” a query by providing another keyword; the matching documents must contain all the keywords**. The customer rules out more sophisticated queries, such as queries that match documents containing any one of their keywords or queries that require the keywords to be adjacent in the document in order for there to be a match. However, such queries are likely in a later release of the product.

**Now we need to consider user and system errors, and also performance:**

The **main performance** issue is how to carry out the queries; the customer wants it to be done expeditiously. This requirement has two implications. First, the program must contain data structures that speed up the process of running a query. Second (and more important) is the question of whether querying requires visiting the Web sites containing the documents. The customer indicates that this should not happen; instead, the query should be based on information already known to the search engine. One implication of this decision is that the collection might not be up to date. A site might have been modified since the search engine was told about it, and queries will not reflect the modifications: they will miss newly added documents or find documents that no longer exist. The customer indicates that this is acceptable but that tracking modifications might be desired in a future release. The customer also indicates that all information about documents should be stored at the search engine, so that if a query matches a document, the user will be able to view the document even if it no longer exists at the site from which it was fetched. One point to note about these decisions is that a trade-off is being made between speed of processing queries versus the space taken for storing documents at the search engine.

Now let’s consider **errors**. There aren’t any interesting system errors: the system has some persistent storage containing information about uninteresting words, but this storage is not modified and the customer is not concerned about media failures. Furthermore, the customer indicates that it is acceptable for the search engine to simply fail if something goes wrong.

There are interesting user errors, however. The user could enter an uninteresting word as a keyword or could enter a word not in any document; the customer indicates that the user should be told about the error in the first case, but that in the second case, the response will simply be an empty set of matches. The user might also present a URL for a site that doesn’t exist, that doesn’t contain documents, or that has already been added to the collection; all of these actions should result in the user being notified of the error. The customer indicates that it is acceptable if a document is found at multiple sites and that, in this case, the document will end up in the collection just once. Two documents are considered to be the same if they have the same title; again, a later release might handle things differently.