Applying Information-Retrieval Methods to Software Reuse: A Case Study

Stierna, Eric J.

Monterey, California. Naval Postgraduate School

Information Processing and Management, Vol. 39, No. 1 (January 2003), 67-74
Applying Information-Retrieval Methods to Software Reuse: A Case Study

Eric J. Stierna and Neil C. Rowe

Code CS/Rp, 833 Dyer Road, U.S. Naval Postgraduate School, Monterey, CA 93943 USA
eric.stierna@sagami-emh1.army.mil, ncrowe@nps.navy.mil, (831) 656-2462

ABSTRACT

Reuse of existing software for new purposes is a key aspect of efficient software engineering. But finding opportunities for reuse can be difficult when building a large software system with pieces of a large previous system. Our approach is to match formal written “requirements” used to define the new software against requirements used to define the old software; requirement pairs with words in common suggest software-reuse opportunities. We explored two matching methodologies that use information-retrieval techniques. We tested our methods by comparing two U.S. military systems, the Aviation Mission Planning System and the Joint Mission Planning System. Our second tool reduced the time to find 50% of the matching requirements by 99.4% compared to manual matching.

Keywords: requirements, matching, information retrieval, document similarity, reuse

This paper appeared in Information Processing and Management, Vol. 39, No. 1 (January 2003), 67-74.

1. Introduction

Good software takes time to develop, and successful software systems must often add new desired capabilities. It helps to reuse
previously written software modules if possible. But identifying the reuse opportunities in large and complex systems, like the most important systems used by the U.S. military, can be very time-consuming. Some automation of the process could be very valuable. The best way to do this is to compare “requirements”, formal written specifications of what the software is supposed to do, which are often precise and succinct (Lim, 1998). Explicit requirements are written before the software is written and are essential for good software development in defining the functionality and interfaces of the software. Carefully written requirements are used by the U.S. government as legal documents in the awarding of contracts. Similarities between systems are often quite clear from inspecting their requirements documents (Jackson, 1995). And often many similarities can be found: Kotonya and Sommerville (1998) claim that 50% of all requirements may be the same for similar software systems in similar domains.

Only a few prototype tools have been developed to aid matching of requirements (Maarek et al, 1991; Maiden and Sutcliffe, 1991; Cybulski, 2001). Some related tools can help, however. Reuse libraries are collections of software resources and related documentation for software development, reuse, and maintenance; manual inspection can suggest reuse opportunities. Domain modeling provide mechanisms for extracting and recording domain knowledge from documents, code, and human experts that can facilitate reuse; examples are the Software Engineering Institute's Feature Oriented Domain Analysis, Organon Motives' Organization Domain Modeling, and the Paramax Systems Corporation's READS tool (Smith, 1993). Formal specifications of software in predicate calculus or formal languages (Clarke et al, 1996) are often similar to requirements, and can be matched using logical reasoning to find reuse opportunities; but formal specifications are even more difficult for humans to write than natural-language requirements, and specification inference is very difficult to automate.

This work examines the effectiveness of information-retrieval methods to find matching English requirements for two software systems. While a certain amount of human judgment is always necessary in determining reuse, our goal is to greatly reduce the necessary effort. We can do that by automatically comparing words between old and new requirements, applying a similarity metric, and presenting the best matches to the software engineer. The matching task is harder than for most document retrieval because requirements tend to be just a few sentences long, and many false matches are possible. Nonetheless, we show that simple methods did reduce the amount of work considerably in experiments with the requirements for two large software systems. More details are provided in (Stierna, 2000).

2. A minimal-automation tool

We first performed a careful matching of two real-world sets of requirements to gain understanding of the matching process, using only minimal software assistance.

2.1. Defining the test domain
For a case study, we examined the requirements of the Aviation Mission Planning System (AMPS) and the Joint Mission Planning System (JMPS). AMPS is a software-based mission-planning tool that automates aviator mission-planning tasks. It can improve battlefield synchronization, intelligence, and command-and-control through communication with the Aviation Tactical Operations Center and the Army Airborne Command and Control System. For aircrews, AMPS generates mission information in hard-copy and electronic formats for upload to aircraft via a Data Transfer System Cartridge. JMPS is a newer and more general system, not yet fully implemented, that provides scaleable mission-planning software for a variety of needs. It supports a wide range of hardware, provides collaborative inter-organization mission planning, and enables information exchange for geographically distributed users. JMPS is a superset of the Air Force, Navy, and Marine Corps mission planning systems; integration of AMPS with JMPS was mandated in a 1998 decision of the U.S. Congress to combine 41 different military mission-planning systems. However, AMPS is not just a subset of JMPS because it contains many features not necessary for JMPS. The hope in comparing AMPS requirements with JMPS requirements is that we can find software reuse opportunities that will eliminate unnecessary development costs during integration of the two systems.

We conducted initial requirements familiarization over four weeks. A domain overview was provided in meetings with a senior project engineer. For further assistance we had a large body of military doctrinal publications such as terminology standards MIL-STD-2525A and various domain-description documents. The written requirements for AMPS comprise an Operational Requirements Document and a System Sub-System Specification (SSS). The JMPS requirements comprise an SSS, an external Interface Requirements Specification, a Concept of Operations, Use Cases, and Scenarios. All requirements were available in electronic format. No System Requirements Specification was available for either system at the time of the study. The primary sources were the AMPS project office, the JMPS project office, and the JMPS development team. The requirements are organized hierarchically but grouping criteria reflect less thematic and topical similarities than managerial organization structures.

2.2. Methods

We focused on the SSS documents in our case study because they contained the most precise descriptions of requirements and appeared the most suitable to automate. The AMPS SSS document (44 pages and 577 requirements) contained fewer requirements than the JMPS SSS document (303 pages and 3538 requirements). We examined similar requirements to gain a sense of the high-level concepts and terms used. We partitioned some long requirements into conceptual blocks in a few cases. We reduced the resulting AMPS requirements to 467 after a second pass to remove requirements without domain-relevant content, as for instance "Reserved" and "The AMPS shall be capable of continuous operation using power from domestic or foreign commercial utility sources of 110-220 volts alternating current (AC)."
A relational database stored requirements and maintained matching information. Figure 1 shows the user interface we developed to support the minimal-automation matching.

![User interface for the minimal-automation matching process.](https://example.com/image.png)

**Figure 1:** User interface for the minimal-automation matching process.

Identifying a match pair involves determining whether one requirement's meaning is fully captured in the other requirement. This involves: (1) reading each requirement in the context of a document's problem domain; (2) evaluating the document's structure and format; (3) partitioning the document into logical sub-domains; (4) evaluating the content of requirements that precede and follow an evaluated requirement; and (5) interpreting the evaluated requirement's actual content.

We chose three to five keywords from each requirement to help in matching, keywords that did not necessarily appear in the requirements itself. To avoid the 467 * 3538 comparisons of every pair of requirements, we only considered matches with a keyword in common. Full (one-to-one) matches were those where one requirement logically entailed the other and vice versa; partial (one-to-
many) matches were those where the logical entailment occurred in only one direction. An example of a full match is:

- **AMPS 3.1.01 AMPS must integrate the applicable DII modules and/or standards into its own structure.**
- **JMPS-090-03100 JMPS shall provide initial Defense Information Infrastructure Common Operating Environment (DII COE) compliance for Windows NT of at least Level-6 and a goal of evolution to compliance at Level-7.**

An example of a partial match is:

- **AMPS 3.1.2.1.5 FLOPPY DISK DRIVE. The AMPS shall contain the necessary hardware and S/W drivers required to be able to read and write files to high density 3.5” FDs via a standard FDD.**
- **JMPS-081-00050 JMPS shall provide the capability to support the physical interfaces that are supported by the Windows NT 4.0 operating system.**

Unmatched requirements usually indicated interfaces to unique systems, transactions, data representations, and devices.

### 2.3. Evaluation

Results were validated by two domain analysts not associated with the matching project. About 25% of the AMPS requirements were slightly modified by the domain analysts after seeing our results. Modifications typically involved decomposition of requirements.

We identified 634 requirements matches, the central question for the AMPS to JMPS migration. 1547 domain keywords were identified, and of the 634 matches, 148 were full and 486 were partial. The total time required was 110.5 hours, for an average time of 3.6 minutes per match pair examined including the time to find the five best matching requirements. The time per AMPS requirement varied from five to thirty minutes, and decreased as we gained experience with the requirements and the document partitioning. Strong matches helped in identifying neighbor matches with larger syntactic and conceptual differences.

The main weakness of our approach was an inability to find all matching pairs in limiting keywords to five per requirement. In random sampling of possible pairs, we found around 2.5 correct matches missed for every one found in the minimal-automation experiment, so we estimate there were a total of 2183 correct matches and we found 29% of them. Finding the remaining 71% would take considerably longer because the remainder were more subtle and required more effort by the analyst to recognize.

### 3. Semi-automated matching
We also developed a more automated tool AMT to further reduce the effort in finding good matches. Its goal was to handle any requirements document, ignoring differences due to grammar, tense, punctuation, and capitalization, and rank matches by quality. We wrote it in Java to support platform independence and permit web-based enhancements.

3.1. Methods

In preprocessing, we removed all non-requirements text within each document (including the table of contents, introduction, glossaries, and appendices), and formalized the requirement labeling (AMPS used codes like 3.2.4.1.2 and JMPS used strings like JMPS-001-00000). Our software AMT then indexed the words of the requirements, ignoring unhelpful words (like "it", "thing", and "since") defined by a list of 613 stop words obtained from a similar list in the MARIE-2 caption-retrieval system (Rowe, 1999) plus 149 additional common domain words (like "JMPS", "require", and "system"). Single characters and single-digit numbers we also ignored. AMT mapped all capitalized words to lower case except for words in upper case since those can have a different meaning in lower case.

AMT then destemmed the words by truncating their suffixes, using our own algorithm related to the algorithm of Porter (1980) but improved with a variety of new rules, a dictionary of 22,000 words, and 1,200 special irregular forms. So for instance our destemmer recognizes "write" and "written", "hold" and "held", "run" and "runnings", "transfer" and "transference", and "analogy" and "analogous" as having the same root form, but not "politics" and "politic" because both are dictionary entries. Destemming reduces the number of distinct words in the requirements (to 3.5% of the original number of words in the text, after also the removal of stop words) and improves matching reliability. AMT then hashed, for each remaining word, the code number of each requirement mentioning it.

To measure the similarity between two requirements, AMT uses the classic cosine formula (Salton and Buckley, 1988). However, interpretation of the weights as word frequencies in the document would unfairly reward words occurring uniformly through all the requirements. So AMT normalizes the word count in a requirement by its count in all requirements. (Normalization with respect to the number of words is automatic in the cosine formula.) Note this normalization is by the raw count in all requirements, not its logarithm as with the popular "TF.IDF" term frequency and inverse document frequency formula, as the rate of occurrence in all requirements is very significant in our application. Our formula for the similarity of two requirements ja and jb, where N is the number of distinct words and p(d,j,i) is the fraction of the number of the occurrences of word i in requirements document d that occur within requirement j, is thus:
These values are calculated for every pair of requirements and stored in sorted order in a table. A software engineer needs to check only the requirements matches with similarity strength above a given minimum threshold, thereby saving time over exhaustive comparison of requirements. Selection of an appropriate threshold needs further analysis over more case studies, but is largely dictated by the number of requirements and the time permitted to study them.

3.2. Evaluation

To evaluate AMT, every pair of the 634 matches found in minimal-automation matching (section 2) that exceeded a similarity threshold in AMT was considered a success. We plotted recall versus precision. Recall we interpreted as the number of successful matches divided by 634. Precision we interpreted as the number of successful matches divided by the number of matches that AMT rated with similarity exceeding a threshold.

AMT found 453 of the 634 manual-keyword matches had at least one word in common in comparing all requirement pairs. Precision was 0.0026 for 90% recall, 0.0080 for 50% recall, and 0.0298 for 10% recall. (Note that this precision estimate just uses the 634, and others of the estimated 2183 correct matches were also retrieved.) To obtain the same 29% recall of minimal-automation matching, a software engineer must examine 6290 matches instead of 2308, but is saved the effort of manually assigning 3-5 keywords to each of 4005 requirements, a reasonable tradeoff. Identifying 149 domain-dependent stopwords definitely helped since performance was 30% worse at 50% recall without them.

Although this precision is poor by the standards of library document retrieval, keyword matching still saved time since there were roughly a million possible matches; so we save 99.4% of the effort in blindly matching all requirements for 50% recall in recommending the high-likelihood matches to a software engineer. The total time AMT took to match requirements was around one hour in our first implementation and 21 minutes in our second on a 300-megahertz workstation; a few minutes of manual work was necessary to set up the databases. This time is modest compared to the manual effort required to accomplish the same thing.

Precision was low because the requirements in the case study were short, and many differences occur in describing the same concept for requirements written at different times, e.g. "acft" for "aircraft", "asset" for "aircraft", "terrain slope shading" for "relief elevation data", "capable of sharing" for "shall be exportable", and "situational data graphics" and "relative positions of all assets." For instance, this match was unfairly low-rated:

\[
S(j_a, j_b) = \frac{\sum_{i=1}^{N} P(a_i, j_a) P(b_i, j_b)}{\sqrt{\left(\sum_{i=1}^{N} [P(a_i, j_a)]^2\right) \times \left(\sum_{i=1}^{N} [P(b_i, j_b)]^2\right)}}
\]
- **AMPS 3.5.3.5: SPECIAL EFFECTS DEPICTION.** The Mission Rehearsal fly through view shall include the following: Terrain slope shading.

- **JMPS-015-15010: JMPS GI&S display capabilities shall enable users to display 3-D perspective views of shaded-relief elevation data and vertical obstructions as viewed from the cockpit.**

Automatic expansion of acronyms to address one aspect of this problem did not help much in experiments. Good thesaurus information of synonyms and superconcepts could help. But other missed matches would require complex knowledge-representation methods from artificial intelligence like plan-based inferencing to recognize:

- **AMPS 3.8.3.2: OUTPUTS FOR OTHER SYSTEMS.** The AMPS shall provide the capability to output the uploaded post-mission data for mission analysis by other users/systems in the electronic media which they require and in printed report formats as designated....

- **JMPS-058-00200: JMPS mission materials generation capabilities shall enable users to display, manipulate, and save materials for output, in common commercial office product formats.**

- **AMPS 3.7.1: COMMUNICATION.** The AMPS shall have a capability to use a TCIM to allow AMPS systems to communicate with IDM equipped Acft or other AMPS located at battalion or brigade TOCs via an interface through the CNR or other appropriate radio. This capability shall allow the planner/user to control the transmit and receive functions of the modem. AMPS shall format the transmit information and format and display received information.

- **JMPS-096-01900: JMPS shall provide message processing services to handle parsing and distribution of fixed-format messages, including military-format messages.**

In light of our results, we explored two refinements of the matching process. One experiment was to index pairs of consecutive words in the requirements, as nominal (noun-noun) compounds are frequent in technical text. To be reasonable, neither word should be a stop word and no punctuation should separate them. We found 894 distinct such pairs in AMPS requirements and 4142 in the JMPS requirements, but only exactly 100 pairs appeared in both AMPS and JMPS requirements. Furthermore, when we examined a random sample of 10 of the 100, we found no correct matches at all in the 41 requirements pairs having one of those word pairs in common. So consecutive-word pairs appeared to be unhelpful clues.

We also explored using the similarities of larger sections (“macrorequirements”) of the requirements documents, which is especially helpful for short requirements whose similarity values were often misleading. The requirements documents were hierarchically...
organized. Multiplying requirement-to-requirement similarities by the summation of the similarities for their immediately enclosing sections of the documents (there were 174 distinct such sections) hurt performance 8% at 50% recall but improved it 23% at 5% recall. So its effect was mixed.

4. Conclusions

Reuse of software modules for new purposes is important in software engineering. Reuse is greatly facilitated if written software requirements can be searched instead of code. We have explored two semi-automated ways to support examining sets of requirements to find potential matches. While this is a difficult matching problem involving short documents, our case study on requirements for two complex real systems showed a large reduction in the time required by a software engineer to locate matching requirements over uninformed manual search, although some human judgement was still necessary. Future work will need to explore synonym data, ontologies, and document organization as further clues.

References


