Advances in Automated and Computer Assisted Coding Software at Statistics Canada

M. J. Wenzowski, Statistics Canada

1. Introduction

Statistics Canada has successfully employed generalized automated coding software in numerous applications since the original release of the ACTR[i] (Automated Coding by Text Recognition) system in 1986. As successful as this software has been, additional requirements have continued to arrive, such that it has become necessary for a replacement to be developed. This paper describes the successor to the current production system.

Designed to operate in all of the major computing environments at Statistics Canada, including hand-held, desktop, server, and mainframe environments, this new release represents a complete re-engineering of the software and its underlying algorithms and components. It contains no proprietary (third party) software, and can be delivered "shrink wrapped" for use at any site, with no additional software purchases or setup required. Coding functions can be embedded within any other application, making the presence of the software completely transparent to the user(s) of the hosting applications. This makes the software particularly well suited for use in data capture operations, including CATI and CAPI. The system also includes an easy to use Graphical User Interface (GUI) which minimizes the burden of maintaining coding dictionaries and coding strategies, and provides a comprehensive automated and computer assisted coding environment for stand-alone coding applications. The kernel of the system is a highly optimized coding engine, designed to concurrently search multiple coding databases, while still providing users with the ability to exercise complete control over the system's text manipulation and recognition processes.

2. System Overview

The next release of ACTR, known as "ACTR version 3" (ACTR v3) is based largely upon the existing production version of ACTR, but contains significant changes in capability. As with its predecessor, ACTR depends upon the pre-definition (entirely under user control) of a database of text and code correlations in order to perform both automated and computer assisted coding. ACTR provides a comprehensive toolset to allow for the creation and maintenance of this *coding database* both from high-volume batch-oriented and GUI-based methods. Once built, the coding database is the object of various searching methods used to attempt to map text to a known code in a computer assisted or automated coding application. In effect, this is the "Text Recognition" referred to in ACTR's name. As described below, the text searching process is quite sophisticated, and is capable of locating not only matching text, but "close matches" as well. As a summary of the overall process, consider the diagram which appears in figure a: the actr process model.

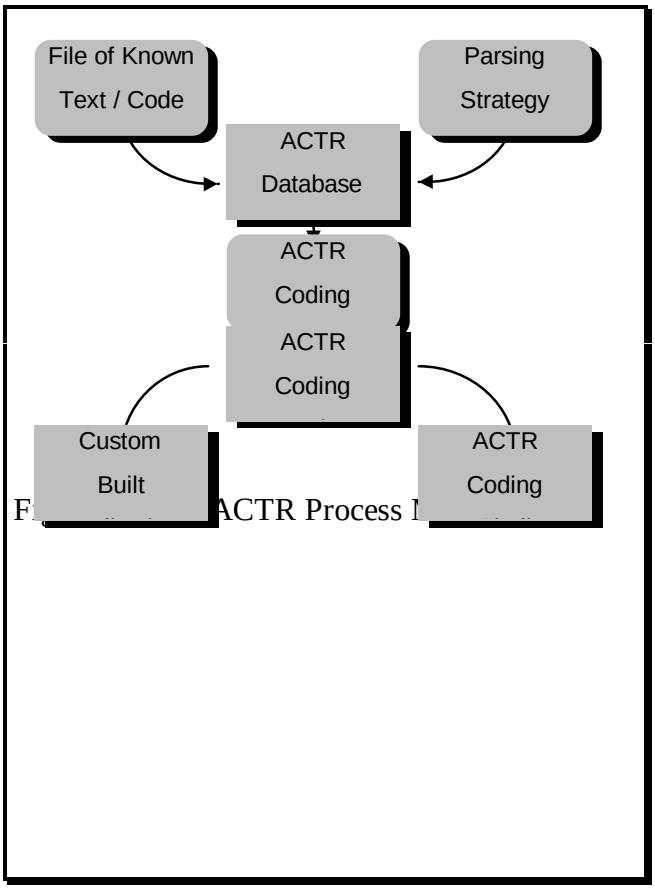


Figure a: ACTR Process Model

The development of an ACTR-based coding application begins with the definition of a set of codes, closely associated with text. These text and code correlations are loaded into an ACTR coding database, and are subsequently used as the subject of a text search during a coding task. When text provided as input to the search is matched with text in the coding database, the previously associated code is returned. Though rather simplistically stated, this is essentially how ACTR functions. Of course, there are numerous complicating factors in searching for matching text - not the least of which is simply defining what we mean by "matching text."

Text recognition requires that sufficiently similar word formats be recognized and consistently mapped to a single word. Examples of variances in word format which a coding application can expect to encounter are: the presence of trivial words; inconsistent use of double letters; and multiple word groupings which may be hyphenated, joined, or entered as separate, multiple words. In ACTR, the function responsible for the standardization of words is known as *parsing*. The flexibility and ease with which users of the software can control the parsing process has played a key role in the widespread success of ACTR within diverse applications, using numerous coding strategies. ACTR has been successfully employed to code a wide variety of variables, including occupations, geographic codes, commodities, marine vessel names and various social and cultural variables. Probably the most successful application to date, in terms of both cost reduction and improved data quality is the use of ACTR in the 1991 Canadian Census of Population[ii]. In recognition that ACTR's user-accessible, highly flexible parsing mechanism has played a key role in its success across diverse applications, ACTR v3 provides extends this mechanism and provides significantly enhanced parsing

capabilities. The parsing process is discussed in greater detail below, in the section entitled: "Text Parsing."

The recognition process continues with the consideration of groups of words, or *phrases*. Examples of problems encountered in this process include missing or extraneous words, and words occurring in an inconsistent order. In the case of a well-tuned, highly optimized coding application, the coding database contains most of the cases expected to be encountered on input. Accordingly, a mechanism must exist for locating these "complete" matches as quickly as possible. This capability is critical in an automated coding function in both high-volume batch processing applications and in high-speed interactive applications, such as CATI and CAPI. The recognition process must also have the ability to locate "partial" matches, or phrases which contain most of the significant words contained in the input text. As expected, performance is critical in batch operations, but since the partial match technique is particularly useful in computer assisted coding functions, performance is again an issue, since interactive response must be as rapid as possible . The text searching process is discussed in greater detail below, in the section entitled: " Search Methods."

3. Text Parsing

As sophisticated as ACTR's searching and matching methods might be, the quality of a text match is almost entirely dependent upon the outcome of the parsing process. Accordingly, the ACTR parser is entirely user-controlled and offers full control of both the parsing data used to effect the parse, and the order in which the parsing steps are applied. A graphic summary of the parsing process appears in figure b: the actr parsing mechanism. It may be useful to reference this diagram while reading the following description of the parsing process.

Ideally, the outcome from parsing is such that any two descriptions with the same words will be identical in their ACTR representation regardless of their syntactical and grammatical differences. For example, the two descriptions: "`computer programmer`," and "`programming of computers`" could, when parsed, result in "`comput program`". How the input descriptions are parsed is controlled by the parsing strategy defined by the user of the software. The strategy may involve the reduction of plural forms, elimination of trivial words, removal of suffixes or any number of other operations.

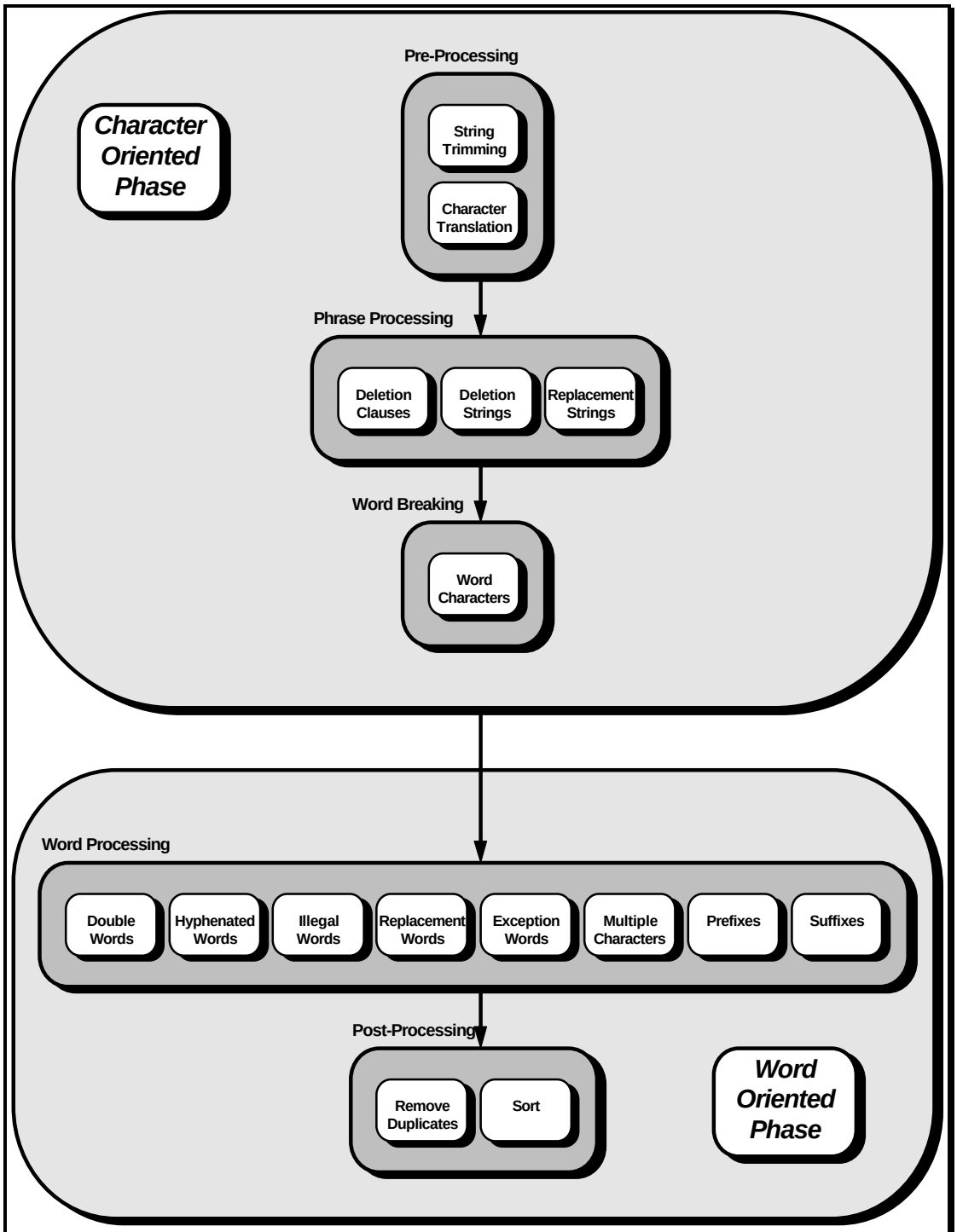


Figure B: The ACTR Parsing Mechanism

The parsing strategy can be easily tailored to suit a particular coding project's requirements. There are three principal mechanisms used to change the parsing strategy:

1. by changing the types of parsing processes that are used,
2. by changing the order in which parsing processes are used, and
3. by changing the parsing data used for parsing processes.

Some research and experimentation is always necessary, in order to determine the best parsing strategy for a particular application. The process of defining a parsing strategy can be complex, especially since changes made within one step or process may affect subsequent steps and processes.

The parsing strategy is comprised of two main *phases*, each with multiple *parsing steps*, acting on their associated *data*. The difference between the phases is the manner in which text is processed. In the "Character Oriented Phase," all text manipulations are performed on a character by character basis, regardless of context or proximity to other characters. Words which may appear in the text are not recognized as such until the second phase of processing, the "Word Oriented Phase."

Within the two parsing phases are five parsing steps. Within each of these steps, users have control over which processes are to be performed, and how this processing is to be performed. This is accomplished through the use of parsing data - changing the parsing data alters the associated process's performance. In addition, within each of the "Phrase Processing" and "Word Processing" steps users may exercise further control by deciding the order in which the processes within these steps will execute.

As with any complex procedure, the parsing process is best understood by individually considering its components. The following sections provide a more in-depth presentation of each of ACTR's parsing components in approximately the same order in which they appear in the diagram.

The "Character Oriented Phase" is composed of the three parsing steps: "Pre-Processing", "Phrase Processing" and "Word Breaking." In these steps, the input text is processed as a continuous stream of characters, and as such, no consideration is given to grammar, punctuation, spelling, word order, or any other contextual information. The central purpose of this phase is to allow for the recognition of particular character sequences, exactly as they appear *in situ*. This can prevent information losses that might occur when the text is subsequently grouped into its constituent words.

The "Pre-Processing" step is composed of two processes: "String Trimming" and "Character Translation," which are always executed in the same order. String trimming is a process which standardizes the input text by eliminating extraneous characters. This may include multiple blanks, tab characters, newline characters, etc. Character translation uses a list of valid word characters and their associated translations to transform the entire input description. This allows control over case sensitivity, and also allows for control of the treatment of accented characters. For example, in converting lower to upper case some applications may wish to transform the lower case letter "é" to an upper case equivalent of "É" (Canadian French), while others may wish to convert it to "E" (International French). Still others may convert both "é" and "É" to "E" and effectively ignore all accents.

The Phrase Processing step is composed of three processes: "Deletion Clauses," "Deletion Strings" and "Replacement Strings." These processes are all optional and if specified, are performed in any order the user chooses. Deletion clauses are a method of delimiting a phrase which should be removed from the input text. By defining a beginning and ending string, any characters enclosed between the strings are removed from the input text. Deletion strings allow for character sequences found at any position in a description to be removed. Replacement Strings are most useful for standardizing abbreviations. This is useful since abbreviations commonly include characters which would normally be processed as word separators.

The Word Breaking step defines a word as any contiguous sequence of characters in an input string which are all members of the set of characters contained in the user controlled word character list. Characters not included in this list are used as word delimiters.

Within the Word Oriented Phase processing is performed on a word by word basis. This phase is composed of two steps: "Word Processing" and "Post-Processing." The order of these steps cannot be changed, but the order of the processes within the steps and the data used by each of the processes can be modified.

The Word Processing step is the first point at which the parser treats the input text as a collection of words. It is composed of eight processes, which can be executed in any order the user chooses. Together, these steps allow for:

- consistent treatment of multiple word occurrences
- consistent treatment of hyphenated words
- removal of words known to be devoid of information content
- replacement of words to ensure consistency (also used to ignore trivial words)
- recognition of words by examining the word root
- reduction of multiple character sequences to a single character instance
- removal of prefixes
- removal of suffixes

The Post Processing step completes the task of parsing. This step is composed of two processes, but their sequence of execution does not affect the form of the parsed output. This step provides for control over multiple word occurrences as well as the final order of the words. For some applications, multiple words are significant while in others they obscure the sense of the text. Similarly, word order may or may not be significant, depending on the application.

The preceding discussion of the ACTR parsing mechanism is merely an outline of its functionality. For complete details the interested reader is directed to the complete set of ACTR documentation[iii].

4. Search Methods

ACTR employs two distinctly different methods for locating matching text. The first of these locates only complete matches, using a technique known as the *Complete Phrase Key* (CPK), while the second, known as the *Indirect Match*, retrieves partial matches, in which both the input text and the database text have at least one word in common.

Ideally, in order to reduce uncertainty and increase coding quality, applications which use ACTR should strive for achieving a state in which their application's coding database contains as many instances of expected text as possible. This being the case, the most efficient manner in which text can be retrieved is to search the database for the input text, precisely as it appears after having been parsed. This requirement is addressed by the CPK. ACTR forms the CPK by accepting the word set returned by the parser, and using a data compression technique, generates a binary string which can be used to provide keyed access into the coding database. To prepare the coding database for CPK searches, all text in the database has a CPK associated with it at load time. In fact, the CPK is used at load time to locate duplicate text in the database, and so control redundancy.

Data compression within the CPK is effected through the location of commonly occurring two and three character groupings, known respectively as digrams and trigrams. These character sequences are replaced by a single byte (eight bits), used by ACTR to represent a

particular digram or trigram instance. Within ACTR, text characters are represented by a single eight bit byte. This representation offers 2^8, or 256 different bit string combinations. However, a typical ACTR application employs a character set which uses only 26 upper case alphabetic characters, 10 numeric characters and the hyphen character. ACTR in turn reserves two additional bytes to represent string termination and word separation. Combined, the total number of used characters is only 39. This means that in the typical case there are 217 bit string combinations which are not used. ACTR associates these unused bit string representations with previously defined, commonly occurring, digrams and trigrams. The result is that the length of the CPK generated by ACTR is typically significantly less than 50% of the original text's length.

The availability of the CPK ensures that ACTR can locate matching text within the coding database in a very efficient manner. It also provides ACTR with the means by which data access can be significantly accelerated, to improve system performance and response times. At the same time, use of the CPK ensures that no information loss occurs as a result of this access optimization.

Typically, Indirect Matching is employed only for those cases in which ACTR is unable to locate a database match using the CPK technique. Basically stated, the technique employed is to find all text with words in common and, through the application of a scoring technique, determine the "closeness" of the matches found. The method is based on the availability of previously calculated weights associated with every word known to the database.

The method used to calculate the weight for a given word is shown in **figure c: word weight calculation** Within that figure, *wordcodes* is the total number of unique codes associated with text in which this particular word occurs, and *totalcodes* is the total number of unique codes defined within the coding database. No pretense is made regarding the validity of the calculation with reference to information theory. Our experience to date indicates that word weights have their greatest usefulness in trimming the search process. Word weights also participate in the text score calculation, used to compute an index of the "closeness" of matches found.

$$WordWeight = 1 - \frac{\log(wordcodes)}{\log(totalcodes)}$$

Figure C: Word Weight Calculation

The technique used to compute a score is shown in **figure d: text score calculation** In this calculation:

- *count(COMMONwords)* is a count of the words in common, between the input text and the database text currently being evaluated
- *count(DBwords)* is a count of the number of words in the database text being compared
- *weight(COMMONwords)* is the word weight of each of the words in common between the input and database text
- *weight(INPUTwords)* is the word weight of each of the known words in the input text

As with word weights, no pretense as to the theoretical correctness of the scoring method is made. It is used as a general heuristic for ranking incomplete matches, and is also used to trim the indirect matching search process. Note that in an effort to make the values more intuitively useful to users, word weights are always in the range of 0..1, and text scores are always in the range of 0..10.

$$a = \frac{2 \times (count(COMMONwords))}{count(DBwords) + count(INPUTwords)}$$

$$b = \frac{\sum weight(COMMONwords)}{\sum weight(INPUTwords)}$$

$$TextScore = 10 \times \left(\frac{a + 2b}{3} \right)$$

Figure D: Text Score Calculation

The search for an incomplete match begins with a database search for all of the words in the input text. For each word found, its associated word weight is retrieved. In descending order by weight, each word is used to find all text in which it occurs. When text is retrieved, a score for the match is computed and compared against threshold values provided by the user, to determine whether or not an acceptable match has been found. In the process, the scoring technique is also used to predict the most optimistic score possible for each word based search iteration. The optimistic score value is compared against the same threshold values provided by the user, to determine whether or not the current iteration of the search should proceed, and in this way serves to reduce search times significantly.

6. Context Switching

ACTR has the ability to assign codes based on multiple database searches. This provides coding applications with a means by which multiple fields may be coded with a single pass. This feature is equally as useful to high volume batch processing applications - in which data file manipulation must be minimized - as it is to interactive processing applications (particularly CATI / CAPI) in which the immediate availability of the respondent requires a single, logical pass through the interview script.

In ACTR terminology, each database opened for matching is referred to as a *context*. An open context associates the database to be used for matching, its parsing strategy and other related matching information. At run time, the application simply passes to ACTR the identity of the context to be used, and ACTR automatically switches its processing, to be associated with the desired context.

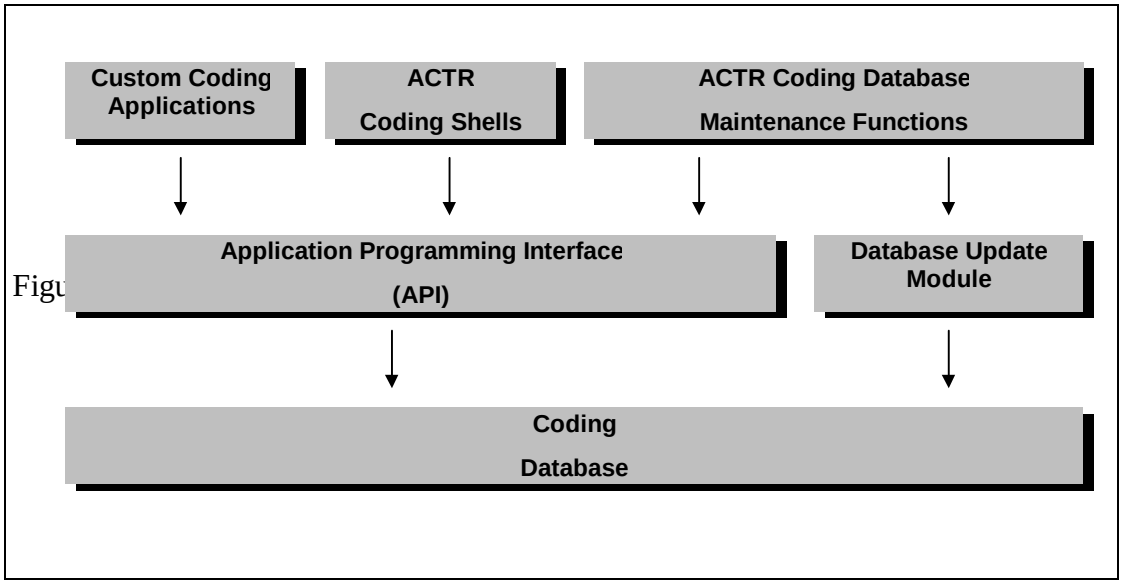
When embedded within a custom application this feature also provides the means by which multi-field dependency coding can be performed. An example of this requirement is the coding of industry and occupation, particularly from data sources submitted by the general population. Coding these variables from the information contained within a single field typically results in low success rates and low quality coding[ii]. By providing the hosting application with multi-field coding, through the use of ACTR contexts, the application developers are provided with the ability to more closely model the manual coding process currently in use.

7. Operating Modes

ACTR is designed to provide coding services in all of the operating modes typically encountered in a large scale statistical agency. The system includes batch processing facilities to provide automated coding services for high volume, unattended coding. These facilities are typically employed through the use of some form of scripting language and are controlled through parameter and control file settings. ACTR also provides a GUI-based interactive facility through which both automated and computer assisted coding can be performed.

Both batch and GUI-based components are referred to a "shells," since they are services which provide the user with a high level interface to lower level coding functions. Users may also call these lower level functions directly, through a custom-developed application. ACTR makes this possible through the provision of an *Application Programming Interface* (API) to the matching functions. The relationships among these components is shown in figure e: actr component model. Note that the Database Maintenance Functions also use the coding API for all database search functions. By ensuring that the API can support all of ACTR's high level processes, custom applications which also use the API can be assured of full functionality.

Coding database maintenance and parsing strategy manipulation functions are provided in both batch and interactive modes, but not through a publicly accessible API. This is an



Figu

intentional design feature. Coding databases tend to be very stable throughout the lifetime of the coding application, with updates typically performed very infrequently and under highly controlled conditions. In coding applications, high-performance coding functions are essential, and in order to achieve this performance, trade-offs are required in the domain of database update mechanisms. Limiting the update mechanism removes the requirement to provide the overhead functions (such as record locking, logging, before and after imaging, etc.) typically associated with multi-user update capability.

8. Summary

The latest release of generalized computer assisted and automated coding software at Statistics Canada has been described in this paper. As this release is still in beta test mode, no performance statistics are yet available. However, a number of significant applications have

already committed to using this new release of ACTR in their production cycle within the next year.

While it is hoped that this paper has provided the reader with a thorough overview of the software, the description is by no means complete. Those who wish to pursue their investigation of this software further are directed to the ACTR documentation set[iii].

References