Quantitative Measurements of Microsoft Source Code and APIs

1. I, and two assistants under my direction, designed and executed procedures to measure the size of the source code for the implementation of the Microsoft operating system and of its APIs. We determined how many lines of code (LOC) are used to implement the Windows XP operating system and the LOC used to make public the Win32 API as represented by the Visual Studio include folder.

2. The following tools were used in this investigation: MKSToolkit Version 6.1; Microsoft Visual Studio 6.0 with no service packs applied; and a custom-built line-of-code counting tool. This tool recursively navigates file system directories processing files based on the presence of an expected file extension. The processing consists of counting total lines, blank lines, a checksum and byte count. The checksum ignores white space, and is intended to detect identical or near-identical source files, for purpose of avoiding double-counting of source code. The tool delegates to the C preprocessor provided with Microsoft Visual C++ the task of determining the number of lines of C/C++ comments. All collected data is stored in a tab delimited text file for post-processing.

3. The LOC tool separately counts blank lines, pure comment lines, and other lines. Only nonblank, noncomment lines of code will be reported here, as this gives the best measure, in my opinion, of the technical complexity of a piece of software.

4. The LOC procedure was executed on the following collections of source code:
   a. The source code for Windows XP as provided by Microsoft;
   b. The source code for the Win32 API as provided in the \VC98\Include within the Visual Studio directory structure.

Processing of File Extension Lists

5. The LOC tool only processes program source files. Not all the files provided to us by Microsoft are actually human-written, machine-translatable files, i.e. program source code. To identify the appropriate set of files to count, we first classified the files by their extension, that is, the letters after the dot in the file name. The goal is to measure all the files that are generated by humans in order to direct the actions of a computer. The procedure for identifying what extensions represent the desired types of files consists of the following:

6. For a given collection of files, determine all the distinct extensions that are present in the collection, the number of files having each extension, and the total number of lines in the files having each extension. This task is performed using the following procedure:

a. By using the utilities provided by the MKS Toolkit, prepare an output file that contains the following for every file in the target directory hierarchy: (i) Filename with extension; (ii) Total lines in the file; and (iii) Byte count of the file.
b. Import the output file into SQL Server using the SQL Server import tool.
c. Determine the number of files, total lines, and total bytes represented by each file extension using a group by SQL statement. The output of the query is inserted into a new "extensions" table.
d. The contents of the extensions table are then exported via the SQL Server export utility (target is Excel 97-2000) to an Excel spreadsheet and the following changes are made to the spreadsheet:
    i. Add a "Process" column. This column will contain 1 to indicate the extension will be processed, 0 to disable processing
    ii. Add a "Reason" column

7. Identify which extensions will be processed by the LOC tool
    a. Identify extensions that, given the context of the file collection, are known, based on the investigator's experience, to be of no interest or consist of contents that cannot be processed by the LOC tool. These extensions are to be excluded from further investigation.
    b. Identify extensions that, given the context of the file collection, are known, based on the investigator's experience, to be of interest and consist of contents that can be processed by the LOC tool. These extensions are marked for processing.
    c. After these two steps, there will still remain dozens of file extensions that may be difficult to classify and which would not in any case contribute significant amounts to the total number of lines of code. We will exclude many of the least significant of these file types, provided that we do not in this way cause more then 10% error in the total measurement. Therefore, based on the sum of bytes for files which are marked for processing in step a., compute 10% of this total byte count. Selecting extensions that are not currently marked starting with the extensions that contribute the lowest percentage to the total byte count, begin totaling the byte counts of these unclassified extensions. Once the 10% has been reached, stop adding to the selection. All extensions in the selection are to be excluded from processing.
    d. For each remaining file extension for which a process yes/no determination has not been made, select the two largest files (based on byte count) having that extension from the source tree for investigation.
        i. These files will be opened and viewed by the investigator.
        ii. The investigator will judge, based on experience, whether the file is human generated, intended to direct a computer, and can be processed by the LOC tool.
        iii. If all three of these criteria are met, in the investigator's opinion, then the file extension is marked for processing, else it is excluded.
        iv. The investigator makes detailed records of files viewed and conclusions made.

Generation of Raw LOC Data

8. The LOC tool is used to generate the raw LOC data by following this procedure: The LOC tool is modified to process the list of extensions marked for processing based on the procedure listed above. The tool is executed on the directory tree that contains the files to process. The output of the LOC tool is saved to a named file for post-processing.

Post-Processing of Line of Code Raw Data

9. The procedure for post-processing the raw line of code data consists of the following:
   a. Import the raw LOC output file into Microsoft SQL Server using the import tool
   b. Alter the resulting table to add the columns Exclude and Reason to the table. (A CategoryID is also added to the table but is not used in this procedure.)
   c. Look for duplicate combinations of path and filename and mark all but the smallest (in code lines) as a duplicate--Exclude = 1, Reason = duplication on import.
   d. Look for duplicate files based on identical checksum and mark all but the smallest (in code lines) as a duplicate--Exclude = 1, Reason = duplication in source tree.

10. There still remain uncategorized (as source or nonsource) several classes of files: the least-significant file extensions totaling at most 10% of the files (measured by byte count) and another group of file extensions totaling an additional 4.2 million lines. Thus, I estimate that the LOC reported in Table 1 for "Full Source Totals", 38 million lines, may be an undercount by as much as 8 million lines.

Results

11. With the LOC data computed and present in the database, an SQL query is performed to compute the total LOC in the XP source tree, the total LOC in the public Win32 API as represented in the header files in the Visual Studio include folder.

Source Collection	Code Lines	Percentage of Total Code Lines in XP Source Tree
Public Interface Totals	443,341	1.16%
Full Source Totals	38,179,316	100%

Table 1 - Windows XP Public Interface Compared to Total XP Implementation. "Code Lines" is nonblank, noncomment lines of source code.