Requirements for Thought.Log—a software to log thoughts

Arun Lakhotia

The Center for Advanced Computer Studies
University of Southwestern Louisiana
Lafayette, LA 70504
(318) 482-6766, -5791 (Fax)
arun@cacs.usl.edu

[The following are "requirements" of *Thought.Log*, a system that would help in keeping track of the "thoughts" of programmers at work. The tone of the document is may imply that such a system actually exists. It doesn't. If you know of some system that meets some or all of these requirements, please let me know. If you would like to develop one, I would like to get in touch with you. AL]

Thought.Log is a software system to log thoughts. It is designed primarily to log the thoughts of one or more programmers at work. The following describes the requirements for such a software system.

1. Thought.Log works as follows:

   a. It monitors the activities of a programmer, classified as follows:

      - No activity at all
      - Browsing of files, without changing their contents
      - Searching for some string
      - Edits/modifications to files
      - Compiling, linking programs; correcting syntax errors
      - Executing programs for testing
      - Executing programs for debugging
      - Writing an e-mail query about the program
      - Responding to an e-mail query about the program
      - Performing support activities (version control, etc.)
      - Writing logs
      - Login
      - Logout

   b. Thought.Log provides mechanisms to support some or all of the activities such that it can monitor which activity is in progress. Otherwise it prompts the user to tell it which activity is in progress.
   c. Thought.Log maintains a time stamped record of the start and end of each activity.
   d. It prompts the programmer about what s/he is doing, planning to do, or has completed doing (as may be appropriate) when

      - transitioning between activities
      - certain prespecified amout of time has been spent since the last log
      - user logs in
      - user logs out
      - user changes the direction of browsing.

e. The Thought.Logs prompts are not very intruisive

   - It does not force a programmer to make a log. It simply reminds the programmer (by beeping or using some visual cue).
   - If the programmer ignores its cue, Thought.Log makes a record of it and prompts him/her sooner the next time around. It may even choose a different form of cue.

f. Whenever there is a significant pause during browsing, or a change of direction of browing, Thought.Log notes the region of the program on screen.
g. Thought.Log records the strings used during searches and records the result of the search.
h. When a program is modified, Thought.Log records the modifications using a version control system, such as CVS. It generates a change log and records it along with the log of thoughts. If the changelog is too big, it maintains only the version numbers of the original and changed code and all the files involved.
i. Thought.Log records the incoming and outgoing e-mails of all the programmers working on the same project in a central archive. In the log of each programmer's thoughts, it maintains a record of the mail (may be just some parts of the header) s/he sends and receives.
j. Thought.Log records the data used for executing a program and also archives the results of the execution.
k. Login is defined as typical computer login, or restart of activities after a long interval of inactivity. At login, Thought.Log asks questions about what's new since the last logout (or cessation of activity). It also asks the programmer about his/her plans for the session, questions/issues that are unclear to him/her.

2. Thought.Log is principally intended to be used in experiments for studying programmers at work. Since confidentiality is important in such studies, Thought.Log provides mechanisms to ascertain it.

   a. It does so by encoding or annotating certain sets of strings/text that may be considered confidential. These strings typically are names of people and organizations associated to the project.
   b. The overall set of confidential strings does not have to remain static, and may changed during or after the course of the project.
   c. However, certain set of strings may be considered strictly confidential, and there status may not be reverted, once defined.
   d. Additionally it provides ways for the programmer to annotate a piece of text as confidential.

3. Thought.Log maintains its database in annotated ASCII (may be using some HTML) and provides various operations on the database, such as:

   a. methods to extract the thought logs for certain time period
   b. methods to extract just the sequence of activities, time stamped or otherwise.
   c. methods to extract logs of only certain types of activities
   d. methods to interleave logs of thoughts with other data such as e-mail exchanges, etc.

Thought.Log — certain design alternatives and contraints

The following are not requirements, but possible design alternatives for realizing Thought.Log. These alternatives may help one visualize how Thought.Log may work and thus help clarify the requirements.

1. The need for detection of an activity may potentially be satisfied as follows.
    a. Interfacing with all the relevant tools (editor, debugger, mail, compiler, more, ..)
    b. Interfacing or instrumenting an environment that integrates all (or most) such tools, such as Brown University's Field.
    c. Developing a loosely integrated environment using *emacs*.
    d. Developing different mechanisms to identify different activities.
        - A specialized shell to identify the use of grep, more, mail, etc. and saving their parameters. This can be customized for individual user.
        - Instrument the editor for identifying some of the activities.
        - Let the programmer identify the remaining activities.
2. The mechanism for detecting activities may be considered separate from the mechanism for actually making logs and prompting the user. The logging mechanism may be developed as a separate utility that runs independently. The utilities detecting changes in activities may communicate with the logging utility over sockets. Such a decomposition will allow paths for incremental development of the whole system... and also permit solutions that may be customized for individual user preferences (thus ensuring ecological validity).
3. The utilities for detecting changes in activity and browsing related details can be developed on *emacs*, probably relatively easily.
4. To synchronize all the logged information, the use of time stamp (date, time) for every entry may require a lot of storage just for the time stamps. An alternative is to use the CVS version numbers as psuedo-time steps. Whenever a log is made, the whole collection of log may be updated (with switches to force the changing of version numbers even if certain configuration items have not changed). This alternative actually may take more space than would keeping time stamps. The only benefit seems to be that it may be easier to synchronize the configuration items, if needed.