IBM Research | CUE | Social Computing Group

Towards Applying Complexity Metrics to Measure Programmer Productivity in HPC

Social Computing Grp. & Web Technologies Grp.,
IBM T.J. Watson Research Center

Catalina Danis, John C Thomas, John T Richards, Cal Swart, Jonathan Brezin, Christine Halverson, Rachel Bellamy, Peter Malkin

10 May 2008

My agenda

- Setting the context
  - Measuring the productivity of HPC programmers
- Introducing the “Complexity Metrics” method
  - Overview
  - CM in the context of IBM’s PERCS measurement approach
- Sources of data for productivity measurement
- The Complexity Metrics method
  - What does it consist in?
  - How we apply it
- Some future directions

Setting the context: Measuring productivity of HPC programmers

- Our task is to evaluate the productivity impact (gain, hopefully!) of programmer tools IBM is developing for the DARPA HPCS program
  - Must address the large scale of (many) HPC codes
  - but also capture events at detailed task level in order to evaluate the impact of tools IBM is developing (e.g.,: IDE, barrier analysis tools, X10 language)
- Productivity assessment is known to be hard
  - Fallback on lines of code, though 180+ others have been proposed
- Strategy of the *"Complexity Metrics"* method
  - Focus on *brief routine tasks* that in combination constitute "*writing code*" (e..g., starting a new software project, getting help on a language feature while writing code, kicking off a performance analysis)
  - Measure programmer behavior doing such tasks using *new and old tools*
  - Build up a picture of overall productivity by drawing on characterizations of task, problem and user types in the HPC domain generated through other means

Other data sources for HPC programmer behavior

- Empirical measurements of programming based studies of programming using "*integrated methodology*" (i.e.,automatic instrumentation supplemented with human observation)
  - Generates data on what *steps* programmers follow to complete a task, *frequency* with which programmers of particular skill level do a particular task, *task error* frequencies ▪
- Field work consisting of interviews with and observations of HPC programmers
  - Cross-validates integrated methodology findings
- Surveys at *"Mission Partner"* work sites and other labs
  - Enables us to characterize the skill level and problem distribution in various labs

Introducing the “Complexity Metrics” method

- Overview
  - An analytic, not empirical approach: We refer to it as a “modeling” approach
    - Focus in on “*routine*” tasks: Characterizable by predictable sequences of actions (e.g., “starting a new software project”, “interacting with documentation while programming”, using a barrier analysis tool to balance synchronization constructs, but not “writing new code”)
    - Define “*representative*” manner of doing task, based on empirical work
    - Define an “*ideal*” programmer with particular error profile: expert, novice
  - Draws on empirical findings in Cognitive Psychology to argue that various dimensions are important for characterizing the difficulty of task completion
  - Asserts that complexity is inversely correlated with productivity
- We leverage IBM’s other PERCS empirical measurement approaches to determine *weight assignments* to calculate overall productivity impact for programmers

The Complexity Metrics method

- Count three elements in human task performance:
  - Number of actions to complete a task, with an action defined to be:
    - CLI: A command with all its parameters
    - GUI: A dialogue box
  - "Working Memory Load": Number of "*data items*" supplied by the programmer and possibly retained: e.g., command name and its arguments, name of a file to open, name of function for which seeking help, directory location where source code may be found
  - And the number of "*context switches*": moving between applications
- Rationale for asserting that these dimensions impact complexity & therefore productivity is based on research findings in Cognitive Psychology
  - Steps generally increase elapsed, chance of errors and opportunities for interruptions (which add time for recovery of state)
  - Retrieval of items from working memory takes longer the more items in memory
  - Context switching requires the use of different conventions and therefore requires time and mental effort to orient to the new context (which may cause increases in time and probability of error)

Some future directions

- Putting the method on better scientific footing
  - We've made simplifying assumptions (all steps are equal; all memory items are of equal difficulty; all context shifts are equal; selecting single path for a task from many that are possible) in order to make the method objective and to enable us to start developing some experience in using it
  - Planning on doing targeted empirical work to define what is a step for users with different levels of expertise; also beginning have frequency data available from empirical studies for task composition

- Extending applicability of method beyond routine work
  - Routine work is -- well -- routine!
    - (also hard to show a 13x improvement)
  - Impact of difficulty of routine work on more cognitively demanding tasks: Is there a carryover effect of making the routine less complex?

Conclusions

- It's a start!
  - Getting experience applying it helps us see where we need to extend it
- Helps us to identify where our tools have difficulties -- want to eliminate "accidental" complexity
- In combination with other measures can speak to important aspects of programming behavior from low level use of tools through overall productivity