Quantifying Program Complexity and Comprehension

Michael E. Hansen
Center for Research in Extreme Scale Technologies
Percepts and Concepts Lab
Indiana University
Bloomington, IN 47405
mihansen@indiana.edu

October 23, 2013

1 Introduction

What makes a program's code complex or difficult to understand? Basili defined software complexity as "a measure of resources expended by a system [human or other] while interacting with a piece of software to perform a given task." [3] For humans engaged in the task of comprehending a program by observing its code, it follows that **code complexity** is a function of cognitive resource expenditure by the observer. The goal of this thesis is to quantify cognitive resource expenditure for a specific programming task: the prediction of a program's printed output. While a plethora of *qualitative* program comprehension models exist, we seek to **develop a quantitative model** from which meaningful measures of code complexity (as defined above) can be derived.

1.1 Of Models and Metrics

This thesis aims to build a bridge between two distinct, but useful, ways of understanding code complexity: via quantitative metrics and qualitative models. On the **quantitative** side, programmers from Academia and industry have created a plethora of *complexity metrics*. These metrics transform a program's source code into easily digestible numbers, and are often used as proxies for how difficult it may be for a programmer to understand the program. Code whose metric values are above agreed-upon thresholds is a prime candidate for refactoring because hard to understand code is less likely to be bug-free.

While practically useful in some industry settings, complexity metrics lack a theoretical foundation in terms of psychology or cognitive science. Recommended thresholds for commonly-used metrics are determined largely based on pop-psychological theories of human memory (e.g., inappropriate generalizations of Miller's Magic Number [10]) and post-hoc regression analyses of

buggy code. The fundamental issue, however, goes beyond overly simplistic theories of memory and cognition: complexity metrics based solely on a program's source code say nothing about the *programmer* reading it. Is the programmer experienced with the programming language, the third-party libraries used, or the surrounding codebase? Has the programmer recently read *this code* or something like it? These and other important questions do not figure into commonly-used size [9, 8], spatial [6], and graph-based source code metrics[7].

The **qualitative** side of code complexity research today is filled with "box and arrow" cognitive models of program comprehension [4]. These models incorporate research from many aspects of cognitive science: psychology, linguistics, computer science, and even neuroscience [12]. They are helpful for making sense of the myriad of results from controlled studies of programmers over the last four decades. Qualitative cognitive models can help emphasize which aspects of cognition are most important for program comprehension, such as language and problem solving skills [14]. Unfortunately, their predictive power is limited. The Stores Model of Code Cognition, for example, emphasizes the importance of the central executive and its relationship to strategic, semantic, and plan knowledge in code problem solving [5]. But, as with source code metrics, predicting whether or not programmer A will successfully comprehend program P is not possible with the model. A *quantitative* model is needed which incorporates aspects of existing *qualitative* models to emulate the process of reading and understanding a program. We believe such a model can be built atop a **cognitive architecture**: software designed to model human perception and cognition [2].

1.2 Proposed Work

Developing a complete model of program comprehension that is sensitive to low level perceptual cues like syntax highlighting as well as high-level abstractions provided by modern programming languages and editors is far beyond the scope of a Ph.D. thesis. Therefore, we must restrict the scope down to something that makes a scientific contribution while still being achievable. To this end, we will only consider the task of **output prediction** for **short**, self-contained **Python** programs.

To inform model development in our restricted problem domain, it is important to know which aspects of a program's source code and the programmer affect how hard the program is to understand. To study this, we will collect output predictions from programmers of varying experience and education levels around the world using Amazon's Mechanical Turk [1]. The programs in these experiments will be readily understood by Python novices, but occasionally break established conventions [15]. By using multiple versions of the same program, we hope to tease out factors affecting comprehension. In addition to output prediction data from Mechanical Turk, we will record gaze data from several dozen programmers as they read and predict output for the same programs. Correlating programmers' eye movements with the intermediary steps of output prediction will give valuable insight into their thought processes without disruption (i.e., via a think-aloud protocol [11]). Both sets of data will provide constraints for model development

and evaluation.

Our model will "read" code that is written in a subset of Python, build an internal (mental) representation of the underlying program, and predict its printed output. By making use of existing models of human vision, memory, etc., the model will be constrained in how it can accomplish this goal. Unlike a compiler or interpreter, it will not parse code into a series of machine instructions and execute them. Instead, a high-level, goal-directed behavior model will coordinate with lower-level behavior models and a set of special modules for human-like declarative memory, perceptual/motor, qualitative spatial reasoning, and inference from domain knowledge.

Like a human programmer, resource constraints and noise will require the model to re-read code as time passes, and to ignore certain details until they become necessary for the task at hand. For example, text on the screen must be attended to before it can be encoded and incorporated into the model's internal representation. Because visual attention is limited to a small region of the screen at a time, and shifts between regions take time, reading in code becomes a bottleneck. The model's declarative memory system will provide simulated speed-ups for frequently and recently attended items, and slow-downs or retrieval failures for unused items. Qualitative spatial reasoning and domain knowledge-based inference will allow for top-down influences on the reading and code understanding processes.

1.2.1 Deliverables

The following artifacts will be produced as part of this thesis:

1. A thorough review of the relevant literature in source code metrics and the psychology of programming
2. An analysis of source code and demographic factors affecting programmer output predictions
3. A methodology/open source library for analyzing programmers' eye movements during output prediction as well as an analysis of collected gaze data
4. A design, prototype, and evaluation of a quantitative process model that predicts visual attention shifts and responses during an output prediction task

1.3 Beyond the Scope

As with any scientific endeavor, we must make some simplifying assumptions to research and model program comprehension. At a high level, our output prediction task is far simpler than what professional programmers experience on a daily basis. While each of our programs can fit on a single screen, it is common for programmers to navigate and understand programs with many thousands of lines of code. For programs of this size, source code highlighting, comments, and code editor features are likely to play a major role in program comprehension. Although our task is basic by comparison, we expect some lessons learned to be applicable to more "real world" scenarios.

Understanding how whitespace affects comprehension and reading efficiency on small programs, for example, could influence style guidelines for large industry codebases.

At a lower level, we assume that the types of data gathered are indicative of the programmer's underlying thought process. For all experiments, we collect keystrokes, response times, and output predictions for each trial (one program). In a pre-experiment survey, participants self-report their level of education and experience with both Python and programming in general. We do not ask participants for verbal reports while reading or responding, and intentionally avoid splitting up the reading and response portions of a trial. When analyzing gaze data, we must assume: (1) the accuracy of the eye-tracking hardware (within some margin of error), and (2) that eye movements correlate well with underlying cognitive processes [13]. Each assumption represents a threat to validity, and must be explicitly kept in mind when interpreting experiment results.

2 Research Plan

A schedule of research activities is proposed below. The bulk of the work will be performed during the period from Fall 2013 to Fall 2014, and a final presentation of results is expected during Spring 2015.

Project	**Planned Dates**	**Status**
Literature review of Psychology of Programming	Spring-Summer 2011	**Complete**
Mechanical Turk and eye-tracking experiments	Spring-Fall 2012	**Complete**
Data analysis and publication of results	Fall 2013-Fall 2014	In Progress
Cognitive model development and possible follow-up experiment	Spring 2014-Spring 2015	Incomplete
Final results	Spring 2015	Incomplete

The cognitive model development project has the highest associated risk because it seeks to quantify a very complex human activity using tools that typically model much simpler behaviors. If sufficient progress cannot be made in modeling programmers' eye movements at the language token level, we can coarsen the analysis to only model visual attention shifts between lines or blocks of code. In addition, the model could parse code into an internal structure via an external process with a fixed time cost. This would remove the need to embed a more complete sub-model of text reading and low-level parsing in the overall model.

We have made contact with potential collaborators at IU, at Southern Illinois University, and at Freie Universität in Berlin. If collaborations can be established successfully, they will provide excellent intermediary steps in the cognitive model development.

References

[1] AMAZON.COM. Amazon Mechanical Turk. `https://www.mturk.com`, Jan 2013.

[2] ANDERSON, J. R. *How can the human mind occur in the physical universe?*, vol. 3. Oxford University Press, USA, 2007.

[3] BASILI, V. Qualitative software complexity models: A summary. *Tutorial on Models and Methods for Software Management and Engineering* (1980).

[4] DÉTIENNE, F., AND BOTT, F. *Software design–cognitive aspects*. Springer Verlag, 2002.

[5] DOUCE, C. The stores model of code cognition.

[6] DOUCE, C., LAYZELL, P., AND BUCKLEY, J. Spatial measures of software complexity.

[7] EL-EMAN, K. Object-oriented metrics: A review of theory and practice. national research council canada. *Institute for Information Technology* (2001).

[8] HALSTEAD, M. H. *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc., 1977.

[9] MCCABE, T. J. A complexity measure. *Software Engineering, IEEE Transactions on*, 4 (1976), 308–320.

[10] MILLER, G. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological review 63*, 2 (1956), 81.

[11] NIELSEN, J., CLEMMENSEN, T., AND YSSING, C. Getting access to what goes on in people's heads?: reflections on the think-aloud technique. In *Proceedings of the second Nordic conference on Human-computer interaction* (2002), ACM, pp. 101–110.

[12] PARNIN, C. A cognitive neuroscience perspective on memory for programming tasks. In *In the Proceedings of the 22nd Annual Meeting of the Psychology of Programming Interest Group (PPIG)* (2010), Citeseer.

[13] RAYNER, K. Eye movements in reading and information processing: 20 years of research. *Psychological bulletin 124*, 3 (1998), 372.

[14] SIEGMUND, J., KÄSTNER, C., APEL, S., PARNIN, C., BETHMANN, A., LEICH, T., SAAKE, G., AND BRECHMANN, A. A new way of measuring program comprehension. Submitted, Sep 2013.

[15] SOLOWAY, E., AND EHRLICH, K. Empirical studies of programming knowledge. *IEEE Transactions on Software Engineering*, 5 (1984), 595–609.