Understanding program complexity: an approach for study

Christopher Douce
Open University
Milton Keynes, UK
c.douce@open.ac.uk

Szonya Durant
Department of Psychology
Royal Holloway University of London
szonya.durant@rhul.ac.uk

Keywords: POP-V.B. recall tasks

Abstract

This short work in progress paper presents a series of experiments that are intended to explore two related research questions. The first research question is whether software complexity metrics have a cognitive basis or are ‘real’ in a psychological sense. The second research question aims to explore in greater detail how programmer memory may relate to measures of spatial software complexity. A small number of metrics are described, followed by a description of two different experiments which are continuing to be designed.

1. Introduction

It has been said that if something can be measured then that something has the potential to be controlled. To overcome the fact that software is such an intangible product, the software engineering community have designed a set of metrics that enable different dimensions of software to be measured. The planned research presented within this paper aims to explore the question of whether the numerical values that are produced by common software metrics have any psychological validity.

The following section briefly describes a number of different software complexity metrics. This is followed by a brief description of a set of experiments that are currently being considered that aim to explore the issue of ‘metric validity’. A discussion section describes some of the perceived challenges and variables that need to be considered, before concluding with a brief summary.

A number of different domains and research papers have inspired this research proposal. The first vector of inspiration has been the area of cognitive neuropsychology, a topic that has been most recently presented during a PPIG workshop by Parnin (2010). Within the broad area of neuropsychology, there are the fields of perception and psychophysics, both of which aim to understand the workings of the human visual system (King, 2009). These related areas may have a lot to say about how human being process and work with complex sets of information. There is much potential that these parallel disciplines might yield findings which may enable us to understand more about our own software development, construction and maintenance challenges.

The PPIG community has been historically interested in subjects such as human memory, debugging and comprehension strategies. Eye tracking equipment has been one of these tools that has been brought to bear on these subjects. The first reference to eye tracking and program comprehension appears to be by Crosby and Stelovsky (1990), who used this technology to uncover how algorithms might be read. More recently there have been publications by Nevalainen and Sajaniemi (2004) who have rigorously evaluated eye tracking tools. There has also been research that aims to explore both program debugging (Bednarik & Tukiainen, 2004) and whether expertise may influence the data that may be captured by eye tracking tools (Bednarik et. al., 2005). Another approach to capturing programmer behaviour without using eye tracking has been demonstrated by Romero et. al. (2005) who used a specially designed tool called the Restricted Focus Viewer.

Research by perception scientists suggests that some eye tracking metrics may relate to the complexity of the stimulus that is presented to participants. With this thought in mind, it is useful to begin to consider how the notion of software complexity is understood by software engineers.

2. Complexity Metrics

The simplest measurement of software complexity is, of course, the number of lines of code a program contains (also known as LOC, or KLOC, for thousands of lines of code). One of the flaws of this measure is that even though a program might have many lines of code, it might not necessarily be very complicated. Subsequently, more sophisticated measures of program complexity have emerged. The most famous ones are by McCabe (1976) and Halstead (1977).

McCabe's cyclometric complexity metric measures the number of independent paths through a program. The more paths there are, the more the programmer has to content with. The Halstead metric, on the other hand, proposes a number of different measures that draws upon a number of different operators and operands that a program contains.

Neither of these early metrics explicitly draws inspiration from the domain of the psychology of programming nor considers the faculties of the human beings who are carrying out software development activities. One of the first attempts to propose metrics that have some psychological validity was a proposal of a suite of spatial complexity metrics (Douce et. al., 1999). Rather than considering the instructions or the number of unique pathways a program contains, spatial metrics draw upon the understanding that software developers make use of their short-term spatial memory when programmers manipulate and work with code. Complexity is therefore considered in terms of the distance between elements of code, providing a different type of measure than the first generation of metrics.

A number of researchers have explored the concept of spatial metrics. The first paper on the work appears to be by Chhabra et. al. (2003), followed by research by Gold and Layzell (2005). More recently, this has been followed by work by Zhang and Baddoo (2007).

Research has found that the earlier metrics such as McCabe correlate well with the simple measurement of lines of code. Zhang and Baddoo have found that the spatial metrics also correlate well with the McCabe metric. These correlation findings implicitly suggest whether there might be scope to uncover whether there are correlations with other ways of measuring complexity. As suggested earlier, perhaps research within the domain of visual perception might be able to help.

3. Proposed Study

Two sets of related experiments are proposed. The first experiment aims to explore if programmers demonstrate behaviourally that complexity exists within code, when faced with code that has been measured as complex by both the McCabe and spatial complexity metrics. The second experiment aims to further explore the validity of the spatial metrics by asking participants to recall what they can remember about different aspects of code they were presented with.

3.1 Participants

Postgraduate students who have knowledge of software development who have some familiarity with the Java language are to be recruited from Royal Holloway, University of London. Participants will not be paid for their participation.

3.2 Materials

Three Java programs, ranging between 100 and 400 lines in length have been developed from a suite of materials that have been created by the first author. The programs are contained in a single file. Each program has been historically designed to demonstrate the use of different elements of an object-oriented language. The entire suite of experimental materials will be made available to with wider PPIG community through the first authors' blog. It is envisaged that each program will be metricated using two sets of software complexity metrics: the McCabe (1977) and the spatial complexity metrics (Douce et. al. , 1999; Gold & Layzell, 2005).

Following each short comprehension exercise, participants will be presented with a series of spatial memory questions. These will take the form of program slices which have been extracted from each of the programs that the participants have been asked to study. Participants will be asked to order the cards based on where the particular fragments of code are located within the program.

3.3 Method

Participants will be asked to read three programs, based on the scenario that is presented below. Please note: the exact wording and sequence of operations is currently being debated and may change.

> Imagine you have just started working as a software developer. A manager has told you to study three short computer programs with a view to understanding how long it might take to change parts of the code. The exact changes that are required are not yet clear.
>
> You will be presented with the three short programs, one after another. Spend however long you wish to study each program. When you have finished studying the program you will then be asked a series of questions about its structure. The first program is to allow you to become familiarised with what occurs within this study.

Each program will be presented to the participants using a simple text editor. Participants will be told that their eye tracking movements will be recorded during the experiment. They will also be told that the experiment is not intended to test themselves, but to help the researchers to find out more about how software developers perceive program complexity. Finally, participants will be told that they will be free to leave at any time, and the experiments should take no more than an hour. Between each program, the program comprehension cards will be presented to the developer by the researcher, and answers recorded.

3.4 Analysis

The precise approaches that are to be used are still currently being debated but the main objective of this research, as described earlier, is to ascertain the validity of software complexity metrics from psychological and perceptual perspectives. The eye tracking hardware will collect a wealth of data, such as fixation time, pupil size and data about which areas of the program code were studied. There will then be a comparison between the complexity metrics that have been produced for each program and the different eye tracking measures to determine the extent to which there might be a correlation, and which of the measures taken by the eye tracking hardware might relate to program complexity.

The spatial memory data can be used in a number of different ways. Firstly, the correctness of the results will be compared with the behavioural data collected by the eye tracking equipment with the intention of learning more about program comprehension, the notion of ‘program space’ and programmer memory. Secondly, the results will be used to understand the extent to which a programmer has formed a successful spatial internal representation of a program. Finally, the correctness of programmer responses may be able to provide new information to allow the exploration and study of the concept of the programming beacon (Brooks, 1983).

4. Discussion

There are a number of challenges that researchers working with this domain face, many of which are described in detail by Brooks (1980) and Sheil (1981). In essence, the challenges can be broadly considered in terms of differences. Research participants can possess very different levels of skills and experience; extreme variation of performance between different programmers is widely recognised, albeit mostly anecdotally. There is also the notion of difference in terms of the experimental materials. One suite of software used as experimental stimuli might be written by another developer or a researcher in a different way. It might be argued that it may be a good idea to move towards a standard set of programs that might be potentially used with different sets of studies to alleviate such variability.

As well as the domain of perception and perceptual complexity, the domain of spatial memory is a related area that may be useful to investigate. As yet, this area has not been extensively studied, but perhaps a literature review of this field may suggest further avenues of research.

5. Summary

This work in progress paper has presented a brief overview of two related experiments which aim to explore whether there is a connection between software complexity metrics and psychological complexity. A set of experimental materials have been prepared and participants will be asked to study a collection of small programs. During the experimental task, eye tracking data will be collected. Following the task, participants will be presented with a short comprehension test which aims to begin to understand the extent to which spatial memory plays a role in program understanding.

6. Acknowledgements

This project has been funded by NESTA Crucible 2009.

7. References

Bednarik, R. & Tukiainen, M. (2004) Visual attention and representation switching in Java program debugging: A study using eye movement tracking. Proceedings of the 16th Annual Workshop of the Psychology of Programming Interest Group.

Brooks, R (1980) Studying programmer behavior experimentally: the problems of proper methodology. Communications of the ACM, Vol 23, Issue 4.

Brooks, R. (1983) Towards a theory of the comprehension of computer programs. International Journal of Man-Machine Studies, Vol 18, No. 6, 543-554

Chhabra, J. K., Aggarwal, K. K. & Singh, Y. (2003) Code and data spatial complexity: two important software understandability measures. Information and Software, Vol 45, 539–546.

Crosby, M. E & Stelovsky, J. (1990) How to we read algorithms? A case study. IEEE Computer, Vol 23, No. 1, 24-35

Douce, C., Layzell, P. J. & Buckley, J. (1999) Spatial measures of software complexity. Proceedings of the 11th Annual Workshop of the Psychology of Programming Interest Group.

Bednarik, R., Myller, N., Sutinen, E. & Tukiainen, M. (2005) Effects of experience on gaze behavior during program animation. Proceedings of the 17th Annual Workshop of the Psychology of Programming Interest Group.

Gold, N. E. & Layzell, P. J. (2005) Spatial complexity metrics: An investigation of utility. IEEE Transactions on Software Engineering, Vol. 32, No. 3.

Halstead, M.H. (1977) Elements of software science. Elsevier.

Jones, S. J. & Burnett, G. E. (2007) Spatial skills and navigation of source code, Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education, pp 231–235.

King, L. A. (2009) Visual navigation patterns and cognitive load, in D.D. Schmorrow et al. (Eds.) Augmented Cognition, HCII 2009, LNAI 5638, pp. 254–259.

McCabe, T.J. (1976) A complexity measure, IEEE Transactions in Software Engineering, 2(4), 308-320.

Nevalainen, S. & Sajaniemi, S. (2004) Comparison of three eye tracking devices in psychology of programming research. Proceedings of the 16th Annual Workshop of the Psychology of Programming Interest Group.

Parnin, C. (2010) A cognitive neuroscience perspective on memory for programming tasks. Proceedings of the 22nd Annual Workshop of the Psychology of Programming Interest Group.

Romero, P, du Boulay, B., Cox, R., Lutz, R & Bryant, S. (2005) Graphical visualisations and debugging: a detailed process analysis. Proceedings of the 17th Annual Workshop of the Psychology of Programming Interest Group.

Sheil, B. A. (1981) The psychological study of programming. ACM Computing Surveys (CSUR), Vol 13, Issue 1.

Zhang, M. & Baddoo, N. (2007) Performance comparison of software complexity metrics in an open source project. Lecture Notes in Computer Science, Volume 4764.