Research Journal

COMP110 - Principles of Computing

1608351

November 27, 2016

1 How do we define a computer?

One of the big debates in computing, is how to define a computer. Is, as has been debated, everything a computer? The answer as far as Horsman et al, are concerned, is no; not everything in the universe is acting as a computer. They argue that declaring everything as a computer, negates the existence of physical computation and that a framework is needed for determining whether a physical system is performing a computation. As such, they propose, that a computer can be defined if all the following are present:

- A strong physical theory of the computer.
- A computational entity.
- A representation relation.
- The ability to encode and decode information.
- One or more fundamental dynamic operations, such as logic gates, which can transfer input states to output states.
- Finally, all these elements need to pass the relevant commuting diagrams proposed in the paper. [1].

2 Reviewing Practices in Computer Programming.

Computing is a fast-changing field, and as such, it is necessary to regularly review best practices to ensure they are still relevant and effective.

One such practice that has been subject to debate, is the use of flowcharts in computer programming. It seems that, as in Mayer's study, whilst flowcharts may be of help with high-level composition, they do not enhance an individuals comprehension [2]. Therefore, it may be possible for individuals to write the program and ones similar, but transferring their knowledge to other coding tasks will be hindered by a lack of understanding at a lower level.

Whilst widely used in both professional and educational programming environments, some researchers have found flowcharts to have little improvement on programming skills. In all five of the experiments performed by B. Shneiderman et al, testing the composition, comprehension, debugging and/or modification skills in university students learning programming, found no significant positive or negative advantages in the use of flowcharts [3]. In one sample, the non-flowchart group outperformed those using flowcharts. It was also observed that, the students rarely made use of the flowcharts, instead working with the program alone. For example, Weinberg explains *'we find no evidence that the original coding plus flow diagrams is any easier to understand than the original coding itself'*[4].

Yet in opposition to such findings, flowcharts have been found effective in other sectors, by helping to separate relevant information from the irrelevant [5]. Further still, there is evidence to support the use of flowcharts over pseudocode in understanding algorithms, *'the more complex the algorithm, the more beneficial structured flowcharts are.'* [6]. As recommended by Sneiderman, performing further study of how structured flowcharts aid, or indeed do not aid, computing professionals may help to determine the relevance of flowcharts to the industry.

A second example of programming practices being subject to review, is the goto

Statement. In 1968, Dijkstra wrote an influential argument against the use of the goto Statement in all high-level programming languages [7]. It is common within the games industry, for teams to change members over-time, so it is important that code is readable and easy to interpret. Dijkstra argues that the use of goto Statements, reduces readability of code, due to our difficulty in understanding dynamically evolving, as opposed to static, relations among objects *'our intellectual powers are rather geared to master static relations and that our powers to visualize processes evolving in time are relatively poorly developed.'* [7]. Going a step further, in 1973, Wulf and Shaw, argued against an over-use use of global variables in larger programs, *'the complexity of the abstraction is directly related to the number of non-local variables used, and we should expect the mental effort necessary to form an abstraction to increase with the number of non-local variables'* [8]. It is generally considered best practice to use global variables sparingly, a change to a global variable can be difficult to locate and therefore difficult to solve.

In the Journal of Robotics and Automation, a 1988 paper is of further help to computer programmers, it proposes an algorithm for calculating the difference between two objects in three-dimensional space, giving programmers a reference when writing collision detection [9].

3 Reviewing Practices in the Game Development Industry.

Following on from best programming practices, in the game development industry, improving development practices are key to ensuring the efficient and successful delivery of working software to clients. Game and software development teams are increasingly adopting the Agile Manifesto, with it's focus on self-organising teams, regular reflection, and communication [10]. Yet, as is to be expected, there are challenges to overcome in the adoption of agile. Research has found certain personality types to have a predisposition towards agile practices, namely extraverted and open individuals [11].

However, those with more introverted and neurotic traits are reportedly less inclined towards the strong focus on accountability and communication found in agile. Yet, a diverse range of personality types can enhance team performance and therefore, it may be beneficial to support those with stronger introverted or neuortic traits, *'The degree of homogeneity of personality of members of the groups used in this study was seen to have a direct bearing on the effectiveness of the groups in producing solutions to problems'* [12]. Such findings, suggest a need to consider personality traits, and the balancing of such traits in teams, when implementing team-driven programming projects.

References

[1] C. Horsman, S. Stepney, R. C. Wagner, and V. Kendon, "When does a physical system compute?" *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 470, no. 2169, 2014. [Online]. Available: http://rspa.royalsocietypublishing.org/content/470/2169/20140182

[2] R. Mayer, "Different problem-solving competencies established in learning computer programming with and without meaningful models." vol. 67, pp. 725–734, 1975.

[3] B. Shneiderman, R. Mayer, D. McKay, and P. Heller, "Experimental investigations of the utility of detailed flowcharts in programming," *Commun. ACM*, vol. 20, no. 6, pp. 373–381, Jun. 1977. [Online]. Available: http://doi.acm.org/10.1145/359605.359610

[4] G. Weinbeg, *The Psychology of Computer Programming.* Dorset House Pub., Jan 1998.

[5] R. Kammann, "The comprehensibility of printed instructions and the flowchart alternative." vol. 17, pp. 183 –191, 1975.

[6] D. A. Scanlan, “Structured flowcharts outperform pseudocode: an experimental comparison,” *IEEE Software*, vol. 6, no. 5, pp. 28–36, Sept 1989.

[7] E. W. Dijkstra, “Letters to the editor: Go to statement considered harmful,” *Commun. ACM*, vol. 11, no. 3, pp. 147–148, Mar. 1968. [Online]. Available: http://doi.acm.org.ezproxy.falmouth.ac.uk/10.1145/362929.362947

[8] W. Wulf and M. Shaw, “Global variable considered harmful,” *SIGPLAN Not.*, vol. 8, no. 2, pp. 28–34, Feb. 1973. [Online]. Available: http://doi.acm.org.ezproxy.falmouth.ac.uk/10.1145/953353.953355

[9] E. Gilbert, D. Johnson, and S. Keerthi, “A fast procedure for computing the distance between complex objects in three space,” in *Proceedings. 1987 IEEE International Conference on Robotics and Automation*, vol. 4, Mar 1987, pp. 1883–1889.

[10] K. Beck et al. (2001) Manifesto for agile software development. [Online]. Available: agilemanifesto.org

[11] D. Bishop and A. Deokar, “Toward an understanding of preference for agile software development methods from a personality theory perspective,” in *2014 47th Hawaii International Conference on System Sciences*, Jan 2014, pp. 4749–4758.

[12] R. Hoffman, “Homogeneity of member personality and its effect on group problem-solving.” vol. 58, pp. 27 – 32, Jan 1959.