COMP110 - Research Journal

Student Number: 1607539

November 26, 2016

Research journal

When does a physical system compute? [1]

Whilst on the surface acknowledging abstract and physical systems as separate entities is a null statement, already implied by their very nature, it sets the groundwork for a system to model the nature of computation. This serves to broaden our view of what a computer can be. Even biological computers can be built from the ground up once appropriate 'logic-gates' or the aptly named 'bio-logic gates' have been identified [2]. In fact, when thinking of a program in its simplest forms: AND, NAND, XOR etc. any system which can resemble these processes can be potentially 'programmed' to execute a computation. Nuclear spins [3], liquid crystal [4], and even DNA [**?**] can have their properties harnessed and manipulated to mimic the principles of 'conventional' computers. When defining unconventional constraint based operations, a computation may be performed providing that there is an underlying physical model sufficiently well characterised [5]. Once we have a system that can be manipulated to solve a problem, that system can function as a computer, "Computing is the automation of our abstractions. We operate by mechanizing our abstractions, abstraction layers and their relationships" [6]. In respects to game development, it is a fascinating concept to one day be able to play a game on a genetically modified lump of slime and being able to install new components by growing them. I wonder how the controls would operate? A slime controller does not sound very appealing whereas the Hollywood visuals of biological interfaces are a stark and suddenly realistic vision of the future. Assuming custom programs could be 'grown' could they evolve? Would they be programmed with a version of a c type code? I suspect that the programs would become more literal and a step back to a visual style of programming like flowcharts could become the norm.

Experimental Investigations of the Utility of Detailed Flowcharts in Programming [7]

There are greatly differing opinions of the effectiveness of flow charts, whilst some believe "the person who cannot flowchart cannot anticipate a problem, analyse the problem, plan the solution, or solve the problem" [7], others criticise them as "a curse," "a space-hogging exercise in drafting" and "a most thoroughly oversold piece of program documentation" [7]. In a controlled study, where two groups which had some knowledge of programming were asked to complete a programming task, one using flowcharts. Subjects who had flowcharts did not perform differently than those who did not have flowcharts [7] questioning the merit of using detailed flowcharts [8], although this has been criticised in its use of such intermediate students and therefore in need of replication using experienced programmers [9]. What can be agreed upon is their usefulness in terms of a teaching tool [10], the methodical and often Boolean aspects closely mirror that of programming computer code although there is a trade-off between teaching flowcharting instead of programming; whether the time be better spent providing more instruction in the language being taught [11]. The obvious solution is to teach flowcharting concurrently with teaching the programming language. The most obvious advantage of flowcharting well is in its use as a visual pseudo code. In the context of game development, it is not uncommon for developers to use their own version of a language or to use a specific unconventional engine. Many people who cannot program or read computer code can understand and follow even the most complicated flow chart with very little training. Giving instructions to complicated processes such as iterations within constraints needed to solve many complicated problems.

A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space [12]

These complicated problems can be complex in the way that they deal with difficult to understand concepts, more than four-dimensional matrix multiplications are an example, or complex in the number of iterations needed to solve. Sometimes both. For example, the detection of collisions and contact gains between moving bodies has the potential to be a very expensive computation. This is because accurate geometric models of realistic physical objects can contain hundreds, thousands, or even tens of thousands of geometric primitives [13]. Game physics rely on algorithms for rigid body, fluid, and cloth [14] and collision detection has been considered a major bottle-neck in computer-simulated environments [15]. Referring back to the Hollywood visions of the future (picture immersion tanks and full body haptic suits) the calculations continue to increase. Consider a haptic device modelling contact constraints, surface shading, friction and texture, remaining stable even as the limits of the renderers capabilities are reached. The computation time is approximately O(log n) where n is the number of polygons, in contrast to the rendering time for a graphic display, where the entire world may be visible at one time, is inherently O(n) [15]. Optimisation is the key to unlocking new technology and the Gilbert-Johnson-Keerthi method [12] and its derivations are used in exciting technologies relating to hand prose estimation [16] and robotics, given that computational efficiency of evolutionary algorithms is better than those of the conventional and mathematical techniques available in literature [17]. However, as complexity and a focus on computational efficiency increases, the readability often decreases. Even the most efficient code must be maintainable.

Go To Statement Considered Harmful [18]

Edsger W. Dijkstra said it best with “The go to statement as it stands is just too primitive; it is too much an invitation to make a mess of one’s program” and perhaps worst with “the observation that the quality of programmers is a decreasing function of the density of go to statements in the programs they produce” [18]. Although he did not just attack the go to instruction for reasons of taste or opinion, but supported his suggested ban by a carefully woven chain of reasoning. One may disagree with some of that argument, but not deny that the conclusion is backed by a well thought-out view of the software development process [19]. Maintaining a high level of rigor with not only the quality of the code but the practice of writing it is essential in writing clear, concise, and perhaps more importantly easily readable and maintainable programs. After all, “Programming is not the same as coding, it entails the many diverse steps of software development. Software process programming should, likewise, not simply be coding, but seemed to entail the many non-coding steps usually associated with application development. Process modelling, testing, and evolution research seems to have borne that out” [20]. Not that Dijkstra did not hold some strong opinions, he is said to have once quipped that “abstract data types are a remarkable theory, whose purpose is to describe stacks”, and that using the word “bug” is a lame attempt by software people to blame someone else by implying that mistakes somehow creep into the software from the outside while the developers are looking elsewhere - as if were not the developers who made the mistakes in the first place. Dijkstra advocated against the use of go to instructions largely to aid readability however I believe this decision should be left to the descretion of the programmer in regards to the program, “without encouraging a programmer to create logical spaghetti” [21]. To an extent a well structured program is a subjective concept. As McCracken has said,“Few people would venture a definition. In fact, it is not clear that there exists a simple definition as yet” [22]. Indeed, “certain go to statements which arise in connection with well-understood transformations are acceptable, provided that the program documentation explains what the transformation was” [21]. We understand complex things by sys- tematically breaking them into successively simpler parts and understanding how these parts fit together locally. Thus, we have different levels of understanding, and each of these levels corresponds to an abstraction of the detail at the level it is composed from [21]. I suppose that this abstraction in terms of readability relates directly to the capability of the reader. I would finish with some further quotes from Dijkstra:

•Write a paper promising salvation, make it a ’structured’ something or a ’virtual’ something, or ’abstract’, ’distributed’ or ’higher-order’ or ’applicative’ and you can almost be certain of having started a new cult.

•The art of programming is the art of organizing complexity, of mastering multitude and avoiding its bastard chaos as effectively as possible.

•If you want more effective programmers, you will discover that they should not waste their time debugging, they should not introduce the bugs to start with.

•Thank goodness we don’t have only serious problems, but ridiculous ones as well.

References

[1] C. Horsman, S. Stepney, R. C. Wagner, and V. Kendon, "When does a physical system compute?" in *Proc. R. Soc. A*, vol. 470, no. 2169. The Royal Society, 2014, p. 20140182.

[2] J. Fisher and T. A. Henzinger, "Executable cell biology," *Nature biotechnology*, vol. 25, no. 11, pp. 1239–1249, 2007.

[3] M. Bechmann, A. Sebald, and S. Stepney, "Boolean logic gate design principles in unconventional computers: an nmr case study," *arXiv preprint arXiv:1109.0918*, 2011.

[4] S. Harding and J. F. Miller, "Evolution in materio: A tone discriminator in liquid crystal," in *Evolutionary Computation, 2004. CEC2004. Congress on*, vol. 2. IEEE, 2004, pp. 1800–1807.

[5] S. Stepney, "Local and global models of physics and computation," *International Journal of General Systems*, vol. 43, no. 7, pp. 673–681, 2014.

[6] J. M. Wing, "Computational thinking and thinking about computing," *Philosophical transactions of the royal society of London A: mathematical, physical and engineering sciences*, vol. 366, no. 1881, pp. 3717–3725, 2008.

[7] B. Shneiderman, R. Mayer, D. McKay, and P. Heller, "Experimental investigations of the utility of detailed flowcharts in programming," *Communications of the ACM*, vol. 20, no. 6, pp. 373–381, 1977.

[8] V. R. Basili, R. W. Selby, and D. H. Hutchens, "Experimentation in software engineering," *IEEE Transactions on software engineering*, no. 7, pp. 733–743, 1986.

[9] R. E. Brooks, "Studying programmer behavior experimentally: the problems of proper methodology," *Communications of the ACM*, vol. 23, no. 4, pp. 207–213, 1980.

[10] R. E. Boyles, "An investigation of the change in motivation of fifth-grade students on writing activities after being taught computer programming using similar teaching strategies," 2014.

[11] T. G. Gill, "Teaching flowcharting with flowc," *Journal of Information Systems Education*, vol. 15, no. 1, p. 65, 2004.

[12] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.

[13] R. Featherstone, *Rigid body dynamics algorithms.* Springer, 2014.

[14] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin *et al.*, "Larrabee: a many-core x86 architecture for visual computing," in *ACM Transactions on Graphics (TOG)*, vol. 27, no. 3. ACM, 2008, p. 18.

[15] J. D. Cohen, M. C. Lin, D. Manocha, and M. Ponamgi, "I-collide: An interactive and exact collision detection system for large-scale environments," in *Proceedings of the 1995 symposium on Interactive 3D graphics.* ACM, 1995, pp. 189–ff.

[16] P. Krejov, A. Gilbert, and R. Bowden, "Combining discriminative and model based approaches for hand pose estimation," in *Automatic Face and Gesture Recognition (FG), 2015 11th IEEE International Conference and Workshops on*, vol. 1. IEEE, 2015, pp. 1–7.

[17] R. Saravanan, S. Ramabalan, C. Balamurugan, and A. Subash, "Evolutionary trajectory planning for an industrial robot," *International Journal of Automation and Computing*, vol. 7, no. 2, pp. 190–198, 2010.

[18] E. W. Dijkstra, "Letters to the editor: go to statement considered harmful," *Communications of the ACM*, vol. 11, no. 3, pp. 147–148, 1968.

[19] B. Meyer, *Object-oriented software construction.* Prentice hall New York, 1988, vol. 2.

[20] L. Osterweil, "Software processes are software too," in *Proceedings of the 9th international conference on Software Engineering.* IEEE Computer Society Press, 1987, pp. 2–13.

[21] D. E. Knuth, "Structured programming with go to statements," *ACM Computing Surveys (CSUR)*, vol. 6, no. 4, pp. 261–301, 1974.

[22] D. McCracken, "Revolution in programming: an overview," in *Classics in software engineering.* Yourdon Press, 1979, pp. 173–177.