# The economic impact of making code portable between computer platforms.

**COMP160 - Software Engineering Essay**

1608557

April 3, 2017

## 1 Introduction

There has been an ongoing demand for portable to code to make it easier and less time consuming to transfer between hardware. This may be for reasons such as robotics and the purpose for each part of hardware is different or hardware and does not support the older software therefore needing new software or re-writing of the old one. This paper aims to explore ongoing problems in this area.

## 2 The need for Portable code

The need for more portable code has been an issue spanning more than 20 years based on my finding from academic papers that are very similar although written at very different times. Colbrook's 'Computing control engineering journal' [5] tells us how the programming language Prelude is for "writing portable parallel programs". This paper from 1992 concludes with more testing of the software is needed for a more reliable outcome. However linking to Vaughan's 'On device abstractions for portable, reusable

robot code' [1] from 2003 they are still looking for a solution to make code more portable for the forever changing hardware purposes and updates.

It is also made clear in [3] " Good software products can have a life of 15 years or more, whereas hardware frequently is changed every 4 years.". I feel that this is part of the problem how hardware is so more frequently updates and software that is optimized for specific hardware is so easily outdated [8][9]. But on the other hand this is why a clear solution for portable code is still an issue and one that has a demand for a solution. I found especially in the robotics industry where hardware is used for very specific purposes that portable code would save a lot of time and money.

I have come across many case studies of researches into making code more portable by using interfaces that support many languages of code [1][5]. One example i found was a program called 'TANGRAM'[9]. This had a fair amount of success with figures showing "code synthesized by TANGRAM for different types and generations of devices achieves no less than 70 percent of the performance of highly optimized vendor libraries". I found this to be a very high percentage so was surprised how this isn't used on a larger scale.[2]

## 3 The money problem

Again we come back to the constant evolution of hardware out-pacing software with different changes due to different capabilities and purposes. One way to prevent loss of funds when altering/re writing code is to sell an abstract Operating System you are using.

One opinion which was against the other was if there is any point in making reusable code at all which is a large discussion in itself with reasons on both sides of the argument. "Good software products can have a life of 15 years or more, whereas hardware frequently is changed every 4 years." [3]. When you have numbers like this it makes you re evaluate if developing portable code is really worth it.

On the other hand though we find studies like this which you can imagine saved a

lot of time and money over a long time. "About 9000 lines of the operating system were implemented in C. The remaining 1000 lines constituted the kernel. The kernel was implemented in assembler and had to be reimplemented for each implementation. About 1000 lines of the C code consisted of device drivers; this code, too, had to be reimplemented each time. However, the remaining 8000 lines of C code remained largely unchanged from implementation to implementation."

## 4 Conclusion

The key points that I can draw from this research are that they are all looking for a solution for more portable code; despite the varied time amounts that these papers were written. This shows the demand and need for an abstract interface that can keep up with current hardware that is evolving every four years. However I have drawn from this research that there are always going to be changes needed for the code but to reduce the amount for example in the C++ case study can save a lot of time and money for employers. This is likely to be an ongoing discussion and a problem that will need more evidence and research upon before an appropriate solution. Some possible solutions have been covered in this paper however they are not 100 percentage reliable and still need further work build upon them.

[1] Vaughan, R.T., Gerkey, B.P. and Howard, A., 2003, October. On device abstractions for portable, reusable robot code. In Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on (Vol. 3, pp. 2421-2427). IEEE.

[2] Pflger, D., Mehl, M., Valentin, J., Lindner, F., Pfander, D., Wagner, S., Graziotin, D. and Wang, Y., 2016. The scalability-efficiency/maintainability-portability trade-off in simulation software engineering: examples and a preliminary systematic literature review. arXiv preprint arXiv:1608.04336.

[3] Schach, S.R., 2002. Object-oriented and classical software engineering (Vol. 6).

New York: McGraw-Hill.

[4] Sametinger, J., 1997. Software engineering with reusable components. Springer Science and Business Media.

[5] Colbrook, A., Weihl, W.E., Brewer, E.A., Dellarocas, C.N., Hsieh, W.C., Joseph, A.D., Waldspurger, C.A. and Wang, P., 1992. Portable software for multiprocessor systems. Computing and Control Engineering Journal, 3(6), pp.275-281.

[6] Ranabahu, A., Maximilien, E.M., Sheth, A. and Thirunarayan, K., 2015. Application Portability in Cloud Computing: An Abstraction-Driven Perspective. IEEE Transactions on Services Computing, 8(6), pp.945-957.

[7] Scacchi, W., 2017. Practices and Technologies in Computer Game Software Engineering. IEEE Software, 34(1), pp.110-116.

[8] Smith, R., Smith, G. and Wardani, A., 2004, December. Software reuse in robotics: Enabling portability in the face of diversity. In Robotics, Automation and Mechatronics, 2004 IEEE Conference on (Vol. 2, pp. 933-938). IEEE.

[9] Chang, L.W., El Hajj, I., Rodrigues, C., Gmez-Luna, J. and Hwu, W.M., 2016, October. Efficient kernel synthesis for performance portable programming. In Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on (pp. 1-13). IEEE.