Can Software Engineering be Taught by Making (and) Music? a proposed empirical study.

Nicolas Gold
University College London, U.K.

Evangelos Himonides
University College London, U.K.

Abstract
The work proposed here centres on materializing core concepts of software engineering (e.g. version management, architecture, the interplay between functional and non-functional requirements, the need to think at both small and large scale simultaneously, user interaction and design constraints, as well as project management) in design thinking through the creation of physical instruments, and their subsequent translation (by introducing the crucial engineering concepts of abstraction and abstract representation) into a combination of software-based digital instruments with physical controllers. The hope is that by doing so, the thinking skills that benefit software engineering (beyond programming) can be developed through physical, not just mental practice. More broadly, students can engage with other aspects of their curriculum (physics, art, design etc) and bring these to bear in an integrated way.

Keywords
making, programming, software engineering, LEGO

Aims
To research ways in which making may be combined with aspects of computer music to create and evaluate secondary-level education resources that introduce fundamental engineering and software engineering concepts at early stage.

There is no doubt that computer music can be used in the context of motivating programming education (e.g the highly-successful SonicPi initiative), however programming is only one part of the broader engineering picture in which software sits.

It is proposed that Lego Technic is used as the foundational material with which to work: children are usually familiar with Lego as a construction tool and system and it has been well-researched as an educational resource. It has been used to construct musical instruments (although there are not many full-scale working examples) and can be easily adapted (through the Mindstorms system or similar) to controller construction with sensors and switches. It is likely that the SonicPi system (or the underlying SuperCollider synth) can be leveraged to provide easy instrument definition and sound production with controllers communicating over local networks, and to introduce programming using the resources that that project has already developed for schools.

The concrete aim is to develop two physical instrument prototypes and corresponding digital/controller versions, create instruction, lesson plans and other educational resources (e.g. 'workshops') to accompany them, to evaluate these in a mainstream secondary school setting, and publish the results. The proposed series of workshops and materials to be developed is envisaged as an 'off-the-shelf' resource for teachers to use either each session individually for particular aspects of curriculum support, or as a coherent programme leading to better student education in computer science and software engineering, delivered through a curriculum founded on computer music.

The intention is that students completing the series would have an appreciation of many underpinning concepts of software development, revealed and demonstrated through physical and subsequently digital means. For example, designing an instrument such as a guitar requires one to consider not just the 'user interface' (balancing norms of playing with creativity), but also the underlying 'architectural' design that must provide sufficient strength not to warp or crack under the tension of strings (and that must be thought about as the instrument is being developed) and functional properties like the transmission and amplification of sound in the body.

There is thus balance between functional and non-functional requirements (along with fundamental lessons to be learned about sound, waves, propagation, amplification, resonance and so forth). Building like this requires design thinking that considers multiple viewpoints simultaneously, experiments with solutions, records things that work and things that don't (version control) and ultimately, after completing the physical instrument, can re-examine and critique it from a new standpoint: the digital instrument. The lessons learned from physical implementation can then be moved into the digital realm: abstracting control and representation of the instrument away from its inherent properties makes some problems easier to solve (e.g. neck tension), and others are new: how should the software be organized? Can we retain history of our development? Others are readdressed in the new context: at what stage do we have to think about fundamental structures? What happens if we don't?

Overall therefore, it is hoped that students come to software development with a physically-developed appreciation of the need for software engineering approaches, and some of the fundamental thinking patterns required.

Acknowledgements

The authors would like to thank the UCL Computer Science Strategic Research Fund for funding this research.

References

1. Kmieć P. S. (2016). *The unofficial LEGO Technic builder's guide*. No Starch Press.
2. Gannod, G. C., Burge, J. E., & Helmick, M. T. (2008, May). Using the inverted classroom to teach software engineering. In *Proceedings of the 30th international conference on Software engineering* (pp. 777-786). ACM.
3. Runeson, P., Host, M., Rainer, A., & Regnell, B. (2012). *Case study research in software engineering: Guidelines and examples*. John Wiley & Sons.
4. *Sonic Pi - The Live Coding Music Synth for Everyone*. (n.d.). Retrieved February 23, 2018, from https://sonic-pi.net/