Teaching Programming Competencies: A Role for Craft Computing?

Tom Crick
Swansea University
Swansea, UK
thomas.crick@swansea.ac.uk

James H. Davenport
Alan Hayes
University of Bath
Bath, UK
{masjhd,ah347}@bath.ac.uk

Tom Prickett
Northumbria University
Newcastle upon Tyne, UK
tom.prickett@northumbria.ac.uk

ABSTRACT

Competency-based education is the recommended paradigm of the ACM/IEEE-CS Computing Curricula 2020 (CC2020) and the Computer Science Curricula 2023 (CS2023) guidelines. Learners apply knowledge, dispositions and skills in a task context as an integral part of their studies is the competency model advocated. While it would be highly unusual to deliver computing-related degree programmes without considering programming in some manner, competency in programming extends beyond simply writing code; indeed, teaching programming is more akin to teaching craft skills than a traditional academic discipline.

CCS CONCEPTS

• **Social and professional topics → Computing education**.

KEYWORDS

Programming, craft computing, software carpentry, competence

ACM Reference Format:
Tom Crick, James H. Davenport, Alan Hayes, and Tom Prickett. 2023. Teaching Programming Competencies: A Role for Craft Computing?. In *The United Kingdom and Ireland Computing Education Research (UKICER) conference (UKICER 2023), September 07–08, 2023, Swansea, Wales Uk*. ACM, New York, NY, USA, 1 page. https://doi.org/10.1145/3610969.3611140

Competency-based education for computing-related degrees is the model recommended by CC2020 and CS2023. The implementation of this approach has been explored by recent ITiCSE Working Groups, with one in 2022 highlighting how a focus on competency may help address the issue of computing graduate employment and underemployment [4]. Learning to program requires the development of craft competencies and deeper understanding beyond the actual writing of code. A learner must develop competencies as a problem analyser/solver, computational thinking, development technical competencies, as well as classical coding [3]. A traditional lecture/workshop approach will not effectively develop these competencies. The apprentice model can be far more effective: a large computing laboratory (100+ seats) with one lecturer being supported by a small team of senior tutors, who are in turn supported by a team of senior undergraduate students can be employed to teach in a practical manner. Software carpentry, codemanship, code literacy, and related sustainable software competencies are developed by such an approach [1, 5]. Here debugging is a craft that must be taught, not a problem that must be bypassed [2]. One advantage with using PhD students as tutors is that they are invariably closer in age to the undergraduates, lending itself to a more peer-to-peer learning experience as opposed to the traditional lecturer/student or master/apprentice model, fostering a more empathetic approach to tutorial delivery. Also, they are sufficiently close to their own undergraduate experience to be able to recall the learning challenges of the respective discipline. This is advantageous as the undergraduate students grow in confidence as they develop their competencies in the subject. We have thus found the following ideas useful: *(i)* As cohort sizes have grown, teaching teams have grown haphazardly. Large teams need structure; *(ii)* We try to assign students to specific groups of seats, each group with its own tutor, encouraging the tutor to engage with each student's learning journey; and *(iii)* The tutors need clear briefing. The temptation is for the tutor to solve the problem; rather, the tutor must be socratic e.g. "*why do you believe the problem is here?*". This poster presents a summary from a number of UK universities on the use of craft computing to teach programming, including the approaches used and the resources required to adequately support it. Craft computing is presented as a good practice model that may be worth considering for broader adoption [2], fostering and promoting software carpentry and "codemanship" as key competencies [1, 3, 6].

UKICER 2023, September 07–08, 2023, Swansea, Wales Uk

ACM ISBN 979-8-4007-0876-3/23/09.
https://doi.org/10.1145/3610969.3611140

REFERENCES

[1] Tom Crick, James H. Davenport, and Alan Hayes. 2015. Innovative Pedagogical Practices in the Craft of Computing. Advance HE. https://www.advance-he.ac.uk/knowledge-hub/innovative-pedagogical-practices-craft-computing.
[2] Quintin Cutts, Maria Kallia, Ruth Anderson, Tom Crick, Marie Devlin, Mohammed Farghally, Claudio Mirolo, Ragnhild Kobro Runde, Otto Seppälä, Jaime Urquiza-Fuentes, and Jan Vahrenhold. 2023. Considering Computing Education in Undergraduate Computer Science Programmes. In *Proc. of ITiCSE'23*. https://doi.org/10.1145/3587103.3594210
[3] James H. Davenport, Alan Hayes, Rachid Hourizi, and Tom Crick. 2016. Innovative Pedagogical Practices in the Craft of Computing. In *Proc. of LaTICE'16*. 115–119. https://doi.org/10.1109/LaTiCE.2016.38
[4] Rajendra K. Raj, John Impagliazzo, Sherif G. Aly, David S. Bowers, Harold Connamacher, Stan Kurkovsky, Bonnie MacKellar, Tom Prickett, and Maíra Marques Samary. 2022. Toward Competency-Based Professional Accreditation in Computing. In *Proc. of ITiCSE-WGR'22*. 1–35. https://doi.org/10.1145/3571785.3574121
[5] Colin C. Venters, Rafael Capilla, Stefanie Betz, Birgit Penzenstadler, Tom Crick, Steve Crouch, Elisa Yumi Nakagawa, Christoph Becker, and Carlos Carrillo. 2018. Software Sustainability: Research and Practice from a Software Architecture Viewpoint. *Journal of Systems and Software* 138 (2018), 174–188. https://doi.org/10.1016/j.jss.2017.12.026
[6] Colin C. Venters, Rafael Capilla, Elisa Yumi Nakagawa, Stefanie Betz, Birgit Penzenstadler, Tom Crick, and Ian Brooks. 2023. Sustainable Software Engineering: Reflections on Advances in Research and Practice. *Information and Software Technology* (2023).