Paper Title: Software Literacy as a framework for tertiary educators

First author:

Title: Dr

Name of first author (first/last): Elaine Khoo

Affiliation: Wilf Malcolm Institute of Educational Research (WMIER), Faculty of Education, University of Waikato, New Zealand

Address: Private Bag 3105, Hamilton 3240

Email: elaine.khoo@waikato.ac.nz

Corresponding author is Author 1? Yes

Second author

Title: Assoc Prof

Name (first/last): Craig Hight

Affiliation: School of Creative Arts, The University of Newcastle, Australia

Address: University Drive, Callaghan, Newcastle, NSW 2300, Australia

Email: craig.hight@newcastle.edu.au

Third author

Title: Prof

Name (first/last): Bronwen Cowie

Affiliation: Wilf Malcolm Institute of Educational Research (WMIER), Faculty of Education, University of Waikato, New Zealand

Address: Private Bag 3105, Hamilton 3240

Email: bronwen.cowie@waikato.ac.nz

Fourth author

Title: Dr

Name (first/last): Rob Torrens

Affiliation: School of Engineering, University of Waikato, New Zealand

Address: Private Bag 3105, Hamilton 3240

Email: rob.torrens@waikato.ac.nz

Submission

Software permeates and impact almost every aspect of our lives today. However not much is understood in terms of how software shapes the teaching and learning of disciplinary knowledge in formal learning contexts. This paper reports on the findings from a two-year funded longitudinal study examining how the notion of *software literacy* is understood, developed and applied in tertiary teaching-learning contexts. We explore the relationship between student success in acquiring software literacy and their broader engagement and understanding of knowledge across different disciplines. A qualitative interpretive methodology framed the study involving two case studies of media studies and engineering students' learning to use discipline-specific software. Student data were collected through class ob-

servations, online surveys and focus group interviews. Findings indicate that a majority of students consider themselves as early adopters of technology, knowledgeable in the affordances and constraints of their disciplinary software, and preferred informal learning strategies to supplement their formal learning of disciplinary software. There was however a lack of student critical awareness of the role of software in shaping their learning of disciplinary knowledge. The findings provide insights into practices relating to tertiary teaching and learning involving software and highlight the significance which programming code may have in the shaping and application of disciplinary knowledge in educational contexts. Implications are offered in terms of tertiary educators' practice and the provision of student learning support.

Introduction

Software is an integral and embedded part of 21st century everyday professional and personal life. Code provides the infrastructure through which everyday life is increasingly 'performed'. Most people develop proficiency with ubiquitous software packages informally through everyday engagement. However, the role and influence of software is largely taken for granted. Software is not neutral; its design and functioning is embedded within particular sociocultural codes of practice and assumptions made by software designers/programmers. For example, the actions of 'copy, cut and paste' are taken for granted as naturalised functions and embedded across different software but poorly understood as tools that shape our engagement with knowledge, culture and society in the 21st century. In tertiary education contexts, educators often assume that students already possess the necessary skills and conceptual frameworks to learn with and through generic software packages to successfully accomplish learning tasks and process, and tend to neglect how the affordances of different software shape the ways students 'perform' the software (Adams, 2006). Emerging evidence internationally and locally indicate a dearth in this digital generation's basic academic literacy skills for successful learning despite their technological competency (Thompson, 2013).

Our research draws from current developments in the emerging field of software studies (Fuller, 2008; Manovich, 2008) to propose the notion of *software literacy* as an inherent and vital but poorly understood part of digital literacy (see example of digital literacy frameworks by Beetham & Sharpe, 2010; JISC, 2014; UNESCO, 2013). We define software literacy as the repertoires of skills and understandings needed for students to be critical and creative users of software packages and systems in a culture now embedded within and increasingly generated through code. Software literacy involves expertise in understanding, applying, problem solving and critiquing software in the pursuit of particular learning and professional goals. It relies on a combination of general competency with software and technologies, together with the ability to undertake more independent (even informal) learning of discipline-specific programmes as and when required.

We hypothesise there exist three progressive tiers of development towards software literacy: 1) a foundational skill level where a learner can use a particular software, 2) an ability to independently troubleshoot and problem solve issues faced when using the software, and finally, 3) the ability to critique the software, including being able to apply such critique to a range of software designed for a similar purpose and to use these understandings for new software learning. The third tier involves the ability to identify affordances and their implications (including the constraints) of particular software and identify ways to both apply and extend its use such that it is relevant and meaningful to a wider range of learning purposes, tasks and contexts. As suggested above, this conceptual model is a response to current limitations in digital literacy frameworks (e.g. Alexander, et al. 2017) which do not go far enough to identify the implications of software (see Khoo, Hight, Torrens, & Cowie, 2017).

The problem being addressed

The research intention was to unpack if and how students develop and use discipline-specific software literacy, understand the influence of specific software applications on the way they make sense of disciplinary knowledge and whether their learning trajectories fit with our hypothesised tiers in the software literacy framework. This aim is translated into the following question:

To what extent and how does student software literacy develop and impact on the teaching and learning of discipline specific software in formal tertiary teaching settings?

The findings can importantly enhance our understanding of how students acquire knowledge and skills to use software and the extent they are able to apply and extend these to successfully learn and act in formal tertiary learning contexts.

Study design/Approach

The research draws from data collected in a two-year longitudinal funded research (Khoo, Hight, Torrens, & Cowie, 2016), to report on the views of participating tertiary media studies and engineering students from a New Zealand university. A qualitative interpretive methodology framed the data collection and analysis in the study. Two case studies of students learning discipline-specific software within two very diverse disciplines of study - media studies and engineering - were developed. An overlapping longitudinal study design (Arzi, 1988) tracked shifts in equivalent student cohorts' software literacy development. For the media studies case (a three year programme), our team tracked one group of students from Year 1 to Year 2; and another group from Year 2 to Year 3 focusing on their learning in papers/coursework involving discipline-specific software (Adobe Creative Suite, Final Cut Pro). Similarly, in engineering (a four year programme), one group of Year 2 students was tracked into Year 3, and another group from Year 3 into Year 4 of their study involving their learning of SolidWorks (a computer-aided design/CAD software). In this paper, we focus only on student perspectives of teaching and learning. Data on an initial baseline study and lecturer perspectives have been reported elsewhere (see Khoo, Hight, Torrens, & Cowie, 2016).

Student data were collected from observations of lectures and laboratory (lab) sessions to understand student learning to use discipline-specific software, online surveys and focus group interviews. The project received human ethical approval and participation was on a voluntary basis. Analysis of the data was underpinned by sociocultural theory which directed attention to the interaction between people, the tools they use to achieve particular purposes and the settings in which the interactions occur (Cole & Engestrom, 1993). Emergent themes from the analysis were identified through a process of inductive reasoning (Braun & Clarke, 2006).

Findings

Four key themes emerged highlighting students': i) general comfort level in engaging with technology; ii) overall preference for and/or reliance on informal learning strategies in acquiring software skills; iii) understanding of core affordances and constraints of disciplinary software applications; and, iv) a relative absence of critical software literacy among students.

i) Student comfort level with technologies

When asked about their general views regarding technology, 38% of students (n=169) reported they usually use technologies when most of their friends do (average across five papers), 35% reported liking new technologies and using them before most people they knew did, and another 18% reported loving new technologies and being among the first to use them. A majority of students (91%) therefore consider themselves early or quite early adopters of new technologies and are comfortable in engaging with new technologies.

ii) Student preference for informal learning strategies in acquiring software skills

Students reported drawing mostly from informal learning resources when learning/acquiring skills to use discipline-specific software (see Figure 1). The three highly valued strategies (combined 'useful', 'very useful' and 'extremely useful') by media studies students were 'Going online to refer to instructions' (91%), "Asking a peer' (86%) and 'Going online to refer to YouTube videos' (86%) as useful to their learning of discipline based software. Engineering students reported 'Asking the teacher' (80%), 'Asking a peer'' (49%) and 'Referring to the course or lab notes' (41%) as their preferred strategies. Possible reasons for Engineering students' valuing asking their lecturer for help before relying on more informal strategies as compared to Media Studies students could be due to the perceived complexity of SolidWorks or less experience at school with CAD software in general.

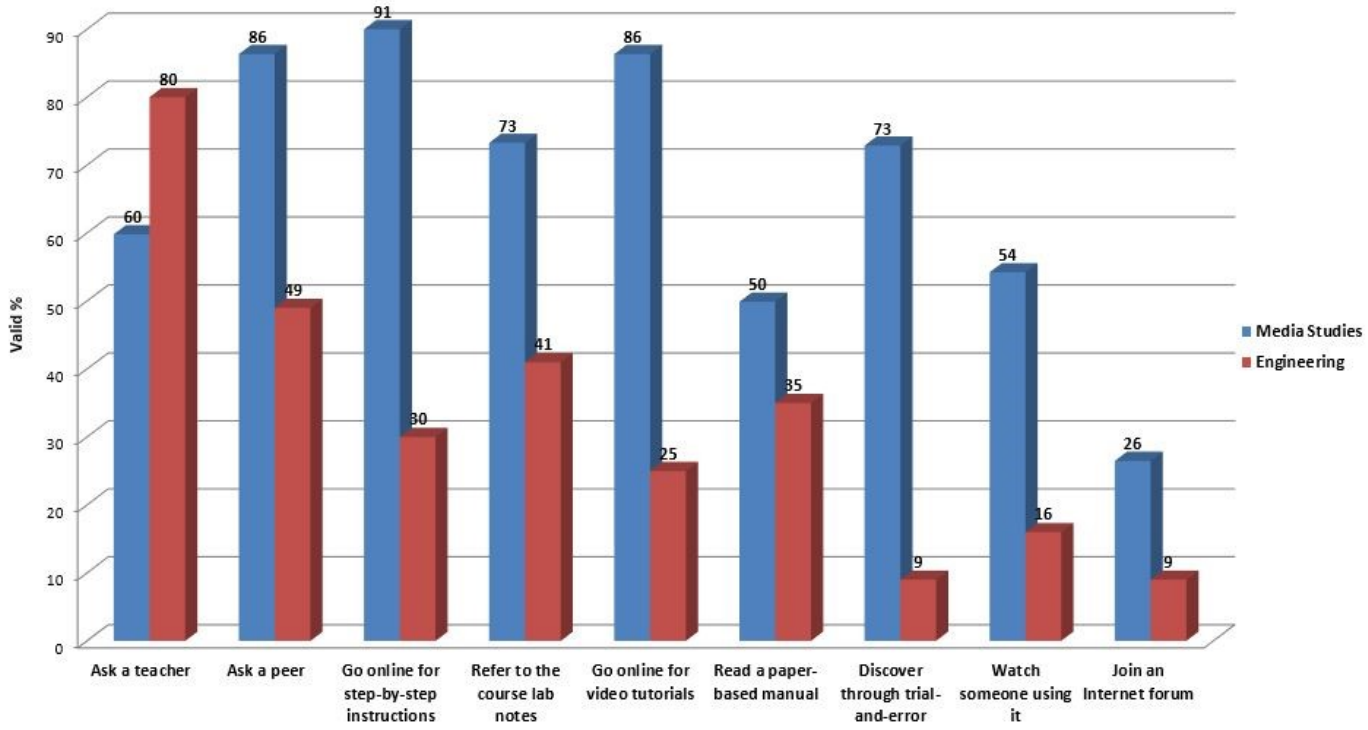

Figure 1. Strategies students used to learn discipline-specific software (collated 'useful', 'very useful' and 'extremely useful')

In focus groups, students highlighted a preference for learning at their own pace; sometimes drawing upon 'more expert' peers or approaching learning collectively when learning professional software. Referring to online resources such as YouTube instructional videos was particularly valuable (as in this representative quote):

> *Trying to follow a software tutor in class is] like watching a YouTube video without pause and rewind…. (Second year Media Studies student)*

Although regular attendance at formal labs is part of coursework for both disciplines informal learning strategies were regularly used to supplement formal lecturer-led sessions.

Many students across disciplines reported the need to invest time and attention when learning discipline-based software to achieve basic competence (and some students explicitly commented that they developed more intensive learning strategies in response). A Media Studies student considered that discipline-specific software (in his case, Final Cut Pro) was sufficiently complex that work done outside the classroom became an important part of learning:

> *… the [lecturers] can give you all the tools but if you're not motivated to do your own experimenting, you're not going to learn the software at all … That's definitely a big part of that is having the time to actually sit down and play around with it yourself. I mean, you can't expect to just be taught the software—it's something that needs time, you've got to learn it, it doesn't happen overnight. (Second year Media studies student)*

This was also the case for engineering students learning to use SolidWorks as one student explained:

> *Cause there's so many tiny little individual parts about understanding SolidWorks that you get past a certain point and suddenly you don't know how to mirror a three-dimensional part (for example). (Second year Engineering student)*

iii) Student understanding of software affordances

When asked regarding their understanding of the affordance and constraints, students across Media Studies and Engineering were able to discern the general affordances of their discipline-based software (tiers 1 to 2 of our hypothesised software literacy model) and to identify the value of these affordances for addressing tasks in their disciplines. For example, the Engineering students noted how SolidWorks affords their addressing of engineering de-

sign issues; it allows them to rotate and manipulate different views of their drawings (81%), easily modify their drawings (79%) and visualise their design drawing to share with others (78%). Similarly, the media studies students reported on the ways their discipline creative software afforded their editing images and sound separately (60%), importing visual and aural elements to combine with their own film footage (59%) and easily manipulating all the elements in a moving image sequence (56%).

Students from both disciplines were also aware of the constraints/limitations of software they were learning and commented on areas such as accessibility (e.g., affordability or incompatibility or crashing issues), time and resourcing demands when learning to use the software, and lack of functionality (e.g., wanting to 3D model in video production software, applying/bringing real images into SolidWorks). While students of both discipline were less likely to identify themselves as 'highly proficient' or 'expert' in using discipline-specific software at the completion of their course, most nevertheless reported confidence in being able to troubleshoot applications (tier 2 software literacy).

iv) Relative absence of critical literacy among students

When asked about their discipline-specific software learning and competency before and after taking coursework, a majority of students reported shifting in their ability to use the software after learning and using it in their course. Based on the categories, 'I would need help', 'I have the basic skills' (tier 1 of our framework), 'I can troubleshoot problems' (tier 2) and 'I can apply this software' (tier 3), students at the start of coursework felt they would need help to use course software, or that they only have the basic skills to use the software. In media studies, at the beginning of coursework, 29% of students reported needing help, 28% felt they had the basic skills, 12% felt they could troubleshoot while 12% could apply the software more extensively. After coursework this shifted to 6% of students needing help, 35% consider they now have the basic skills, 28% could troubleshoot problems and another 29% could apply the software more widely. Similarly in engineering prior to coursework, 52% felt they needed help initially with SolidWorks, 39% thought they had the basic skills, 6% could troubleshoot problems encountered while only 3% could apply SolidWorks to a wide range of tasks. After coursework, 1% reported needing help, 45% felt they had the basic skills, 37% could troubleshoot issues, and another 16% thought they could apply SolidWorks more extensively.

Most students however had difficulty identifying core disciplinary ideas embedded within software, or felt they were unable to critique the software they were using. Very few students in Engineering discussed how SolidWorks shaped their disciplinary knowledge (a key part of software literacy) as with students in Media Studies. Triangulation of data sources suggested that the few students who reported being at tier 3 were in most cases already competent on entry to the course.

Students interestingly highlighted that their learning of discipline-specific software is facilitated by having prior engagement with software or artefacts that had a similar conceptual basis and provided a pathway for them to engage with new and more advanced software learning. Media studies students for example reported that prior experience with Photoshop made easier to pick up the skills to use other media software; while engineering students similarly alluded to the role of introductory design software such as Google SketchUp.

Students also proposed ways lecturers could approach the teaching of discipline-based software in order to enhance their appreciation of the socioculturally and historically relevant disciplinary ideas embodied within the software including their authentic applications in real-life. In media studies, students raised the need to understand the broader contextual/conceptual framework behind the design of a software application:

> *Like in Final Cut Pro, words like "bins" and other words they go back in history to actual bins that you put film footage into and the cutter will bring them out and cut them. I think that the history of editing and why those terms are used and giving them a bigger picture might just help them realise the terms. […] it's just that deeper knowledge that's very shallow when you're coming into software if you don't know the history of the industry that goes behind it (First-year media studies student).*

While engineering students highlighted raising awareness of the software's possibilities:

> *I think what would be cool is if we had case studies or something; just some problems in class we could work through, the teacher could go through, like, "this is something that you may encounter while you're doing CAD, this is how we've gone about it, you could do it your way but this is the procedure we've used" (Second-year engineering student).*

Discussion and conclusion

The study findings evidenced the existence of our proposed three-tier software literacy framework. Most students were comfortable to achieve tiers 1 and 2 in perceiving themselves to be early adopters of technologies, able and willing to engage in informal learning strategies to supplement formal lab based teaching of discipline-based software and could troubleshoot issues encountered with software. Our framework offers a conceptual tool for practitioners in understanding the role of troubleshooting as an important development stage in learning with and through software. However our findings affirmed a lack of students achieving tier 3; that is developing a critical awareness of the influence of software in shaping disciplinary knowledge and extending this critique to other software applications. As such, some implications for tertiary educators teaching software include:

- Lecturers can tap into students' informal learning strategies and resources as a supplementary to and at times above formal strategies to learn discipline-based software. This ensures diverse students' learning needs can be met through multiple pathways for exploring a software's affordances.
- Lecturers need to explicitly teach and model software critique if they wish to foster this capacity and/or make this possibility known to students. Students' superficial critique of software challenges current assumptions of today's digitally literate generation; critical awareness is not necessarily due to familiarity with and regular use of software.
- Lecturers can direct attention to formatively assessing students' initial software literacy and adapting teaching activities accordingly as there is an advantage of more advanced software learning for students with prior engagement with other similar software. Lecturers adopting a range of teaching approaches (formal and informal) and being flexible to address diverse learning needs will be valuable in supporting student learning.

As digital technologies and the software embedded within them become increasingly pivotal in everyday life, it is crucial that we be aware of their influences on the ways we come to know and understand disciplinary knowledge in tertiary teaching-learning contexts. This research goes some way towards addressing these issues.

Acknowledgements

The authors gratefully acknowledge funding support from the Teaching and Learning Research Initiative, New Zealand Council for Educational Research, Wellington, New Zealand.

References

Adams, C. (2006). PowerPoint, habits of mind, and classroom culture. *Journal of Curriculum Studies*, *38*(4), 389–411.

Alexander, B., Adams Becker, S., Cummins, M., & Hall Giesinger, C. (2017). Digital Literacy in Higher Education, Part II: An NMC Horizon Project Strategic Brief (Volume 3.4). Austin, Texas: The New Media Consortium. Retrieved from https://blog.stcloudstate.edu/ims/files/2017/08/2017-nmc-strategic-brief-digital-literacy-in-higher-education-II-ycykt3.pdf

Arzi, H. J. (1988). From short-to long-term: Studying science education longitudinally. *Studies in Science Education, 15*(1), 17–53.

Beetham, H. & Sharpe, R. (2010). Digital literacy framework. Retrieved from http://jiscdesignstudio.pbworks.com/w/page/46740204/Digital%20literacy%20framework

Braun, V. & Clarke, V. (2006). Using thematic analysis in psychology. Qualitative Research in Psychology, 3(2), 77-101

Cole, M. & Engeström, Y. (1993). A cultural-historical approach to distributed cognition. In G. Salomon (Ed.), Distributed cognitions, psychological and educational considerations (pp.1-46). Cambridge: Cambridge University Press.

Fuller, M. (2008). Software studies: A lexicon. Cambridge, MA: The MIT Press.

JISC (2014). *Developing digital literacies*. Retrieved from https://www.jisc.ac.uk/full-guide/developing-digital-literacies

Khoo, E., Hight, C., Torrens, R., & Cowie, B. (2017). *Software Literacy: Education and Beyond*. Singapore: Springer.

Khoo, E., Hight, C., Torrens, R., & Cowie, B. (2016). *Copy, cut and Paste: How does this shape what we know?* Final report. Wellington: Teaching and Learning Research Initiative. Available at http://www.tlri.org.nz/tlri-research/research-completed/post-school-sector/copy-cut-and-paste-how-does-shape-what-we-know

Manovich, L. (2008) Software takes *command*. Cambridge, MA: The MIT Press.

Thompson, P. (2013). The digital natives as learners: Technology use patterns and approaches to learning. *Computers & Education, 65*, 12-33.

United Nations Educational, Scientific, and Cultural Organization (UNESCO). (2013). *Global Media and Information Literacy (MIL) Assessment Framework: Country Readiness and Competencies.* Retrieved from: http://unesdoc.unesco.org/images/0022/002246/224655e.pdf