The 'Art' of Programming: Exploring Student Conceptions of Programming through the Use of Drawing Methodology

Adon Christian Michael Moskal
Otago Polytechnic
Dunedin
New Zealand
adon.moskal@op.ac.nz

Joy Gasson
Otago Polytechnic
Dunedin
New Zealand
joy.gasson@op.ac.nz

Dale Parsons
Otago Polytechnic
Dunedin
New Zealand
dale.parsons@op.ac.nz

ABSTRACT

In this exploratory study, we analysed 396 drawings by first-year programming students in response to the question "what does programming mean to you". We were surprised by the level of care that students gave to their drawings, and we were confronted by the degree of emotion contained within the drawings. To date, few studies have focused specifically on programming students' emotional reactions to their learning experiences. Here, we analysed our student drawings as 'group data', taking note of recurring artefacts, actors, activities, aspirations and affect across the entire dataset. The observed patterns noted in the drawings raised questions around how students conceptualise programming, both as a subject and potential future profession. As contributions to the field, we: (1) discuss the potential of drawing as a research methodology for computer science; (2) present our findings and observations; and (3) suggest how this type of data could be used to better inform teaching practice in novice programming courses.

CCS CONCEPTS

• Social and professional topics~Computer science education • Social and professional topics~CS1

KEYWORDS

Programming education; drawing methodology; student affect

ACM Reference format:

A. C. M. Moskal, J. Gasson, and D. Parsons. 2017. The 'art' of programming: exploring student conceptions of programming through the use of drawing methodology. In *Proceedings of ICER '17, Tacoma, WA, USA, August 2017*, 9 pages.
DOI: 10 http://dx.doi.org/10.1145/3105726.3106170

1 INTRODUCTION

In 1974, accepting his Turing Award for contributions to the field of computing, Donald Knuth delivered a lecture questioning the classification of programming as an art or a science. "Programming," he said, "can give us both intellectual and emotional satisfaction, because it is a real achievement to master complexity and to establish a system of consistent rules." [1:670] We believe it is exactly this juxtaposition of science and art, logic and emotion, that comes to bear when novices learn programming.

In this exploratory study, we used a novel approach to investigate the learning experiences of first-year programming students—drawing methodology. We begin with an overview of the challenges of teaching novice programmers, explain our rationale for using drawing as our research methodology, and finally discuss our findings and wider implications. The aims of this paper are twofold: (1) to explore generally the potential of using drawing data in computer education research; and (2) to specifically use drawing methodology to investigate the programming experiences of our first-year students, to see if we can gain insights for improving our teaching practices.

2 TEACHING NOVICE PROGRAMMERS

The challenges inherent in teaching novices programming are well-documented [2, 3]. Students find it hard to grasp fundamental concepts, and get frustrated when programs do not work as planned or at all. As educators, we are constantly seeking new ways to improve our teaching methods, curriculum design, and ultimately the learning experiences of our students.
Any investigation into the teaching and learning of programming should consider the interplay between three entities: (1) the course content, (2) the teacher, and (3) the student, each of which can impact on the success of the novice to grasp introductory programming concepts [4].

First, thinking of the content of introductory programming courses, it is difficult for many students to learn the abstract concepts of programming (such as conditionals and loops), wrestle with the syntactic constraints of different languages, and constantly apply their emerging knowledge to new and unfamiliar problems, making rote learning a challenging task [5]. As well as the difficulties in learning explicit coding skills, introductory programming courses can also address peripheral subjects, such as logic thinking and problem-solving, 'soft skill' development (such as communication and teamwork, typically in the form of pair programming exercises), mathematics, and business/client considerations.

Second, there is considerable variation in how teachers teach introductory programming. For example, Pears, et al. [6] present a comprehensive review of the factors that can vary from teaching context to teaching context, including choice of teaching strategy, programming language, or support tools (such as an Integrated Development Environment, or IDE).

Third, the personal qualities of the student can dramatically affect their performance in an introductory programming course. Many studies have explored students' personal attributes to determine predictors of student success in introductory programming courses [7]. As well as being cognitively challenging, introductory programming has also been shown to carry a high emotional load; for example, Kinnunen and Simon [8] explore the effect that student experiences of programming (positive and negative) can have on their self-efficacy. Gasson, Parsons, Wood and Haden [9] have also shown a clear link between student affect and performance in first-year programming papers.

Student conceptions and misconceptions of programming can also contribute to their overall success [10], and it can often be difficult for computer science educators to comprehend where these conceptions come from [11]. Student conceptions of computer science have been examined from a number of perspectives: for example, Liang, Su and Tsai [12] present an assessment of Taiwanese college students' conception of and approaches to learning computer science; Stamouli and Huggard [13] explore undergraduate computing students' perceptions of program 'correctness'; Krpan, Mladenović and Rosić [14] look at the relationship between novice programmers' perceptions and success in introductory programming courses; and Eckerdal, Thuné and Berglund [15] report on first year computing students' understandings of what it means to 'learn to program'.

3 DRAWING AS A RESEARCH METHODOLOGY

"If I could say it in words, there would be no reason to paint."
— Edward Hopper

Answering Fincher, Tenenberg and Robins' [16] call that computer science education needs to augment its traditional research methods, we utilised drawing as our research approach to investigate novice programmers' conceptions of programming. Similar visualisation techniques have been employed in computer science education research, for instance Hübscher-Younger and Narayanan's [17] novel approach to teaching algorithms; however, we could not find any studies in computer education specifically employing drawing as their research approach.

Drawing as a research method(ology) has been used extensively in the social sciences, particularly in the areas of psychology, and then particularly with children [18, 19]. Primarily, the benefits include being able to express one's self without having to rely on words—e.g. for children with an underdeveloped vocabulary, or for comparisons between international contexts—or to represent things which are difficult to convey in words (for instance, motion). For our purposes, while our students are not children, as novices they are likely to have a similarly underdeveloped disciplinary and academic vocabulary [20, 21]. And, as Guillemin [22:275] notes, "Drawings … are about how people see the world in both its simplicities and its complexities. Drawings are intricately bound up with power relations, social experiences, and technological interactions".

Within primary and secondary school education, drawing has been utilised as a research methodology to explore students' conceptions of various subjects. For example, Selwyn, Boraschi and Özkula [23] asked primary school students to express their conceptions of information and communications technologies in schools through drawings. More recently, drawing has also been used to explore secondary student conceptions of 'learning' [24, 25]. In both examples, the authors espouse the benefits of using drawings to allow their participants to express themselves in new and different ways than traditional text-based approaches (such as interviews or questionnaires).

Drawing as a research methodology has also been used in higher education. For example, Sim [26] used 'participatory drawing' for research into PhD students' conceptions of their doctoral research process—students were invited to depict their research process in diagrammatic form, with the diagrams being used later as stimulus for further discussion. Similar to the primary and secondary school examples above, the use of drawing (in this case, coupled with participant discussions) "offered the opportunity for participants to convey deeper and more varied internal representations or meanings." Köse [27] also used drawing with students at a Turkish university to uncover their misconceptions about science.

4 METHOD

This study took place at Otago Polytechnic, in Dunedin, New Zealand, within the Bachelor of Information Technology (BIT) degree. We analysed 396 first-year student drawings in response to the question "Draw, sketch, illustrate, paint, depict or otherwise portray what programming means to you." 206 drawings were from Programming 1, and 190 drawings were from Programming 2 (henceforth, P1 and P2 respectively). P1 focuses on programming fundamentals (e.g. variable manipulation, and flow of control) and patterns, while P2 introduces students to Object Oriented (OO) concepts, such as classes, objects, inheritance and polymorphism. The drawings were produced at the end of the final exam for each class, the last question on the exam paper inviting the students to draw their representations of programming. The drawing data represents 6 years' worth of data, collected since 2010.

Our method for analysing the drawings was largely informed by Selwyn, Boraschi and Özkula [23], who used drawing as a research methodology to investigate primary students' conceptions of ICT in schools. Following their lead, we analysed our drawings as group data, looking for patterns across the entire dataset, rather than trying to unpack what individual drawings might be saying about specific students. We analysed the drawings and applied codes where the content or items were 'meaningful' to the concept of 'programming'; drawings could have multiple codes, but individual codes were only attributed once to each drawing. Student responses that were purely text-based with no pictorial component were excluded from the analysis.

The entire dataset of drawings was analysed and coded collaboratively by the three researchers. Researchers B and C are lecturers in the first year of the BIT degree, and have each taught on the degree programme for over 20 years; researcher A is a lecturer in the second and third year of the degree, and has been with the department for less than a year. As such, researchers B and C offered 'inside' or emic perspectives on the drawings, while researcher A offered an 'outside' or etic perspective.

We chose to collaborate on the coding process to capitalise on these different perspectives [28], discussing the codes, negotiating meanings, and socially constructing our interpretations. As Nielsen [29:2] argues:

Collaboration in qualitative data analysis runs the risk of being reduced to comparing their individual analyses and thus eliminating the opportunities in collaborative analysis. Collaborative analysis performed by a (small) group of researchers may well create the advantage to the researchers informing, influencing, and justifying through a dialogue with each other on how they can arrive at a joint analysis. Differences in perceiving the data can then be viewed as an opportunity for learning rather than merely a source of reduced reliability.

We used the four themes determined by Selwyn, Boraschi and Özkula [23] to guide our coding:

1. *artefacts*: programming objects or 'concepts' represented in the drawings;
2. *actors*: people represented in the drawings;
3. *activities*: what the people appear to be doing/their actions in relation to programming;
4. and *aspirations*: the aims/goals that the drawings seemed to be suggesting (with regard to programming).

Through our coding process, we also came up with a fifth category, *affect*, or the emotions about programming that the drawings appeared to be conveying. Each theme was further divided into subthemes as necessary. The findings of our coding process are described in the next section.

5 FINDINGS

We will now outline the primary findings of the coding process, according to theme. Example student drawings from each category are shown below, and further examples can be viewed at http://bit.ly/2u1ZB3D.

5.1 Artefacts

Out of the 396 drawings analysed, 47% (n=185) contained programming artefacts, such as depictions of computers or snippets of pseudocode. Personal computer components were the most frequently depicted, such as monitors (24%, n=96), peripherals (e.g. mice and keyboards; 20%, n=81), and CPU units (9%, n=35); laptop or tablet devices were depicted least frequently (3%, n=13). Other programming artefacts such as snippets of code (7%, n=29), blocks of logical pseudocode (6%, n=25), and logic/flow diagrams (3%, n=13) also featured throughout the dataset. Examples of drawings containing programming artefacts are shown in Figure 1.

5.2 Actors

Students most frequently depicted no actors in their drawings (52%, n=207), followed by one actor only being shown (44%, n=173). Drawings showing more than one actor were far less frequent (4%, n=16). Figure 2 shows an example of each of these categorisations—no actor depicted, one actor depicted, and more than one actor depicted.

5.3 Activities

Only two activities pertaining to 'programming' were identified in the student drawings: 10% (n=41) contained depictions of actors programming or writing code (as discerned by representations of 'hands on keyboard'), and 3% (n=11) showed some representation of problem solving (e.g., puzzles being solved). Figure 3 shows examples of these activities.

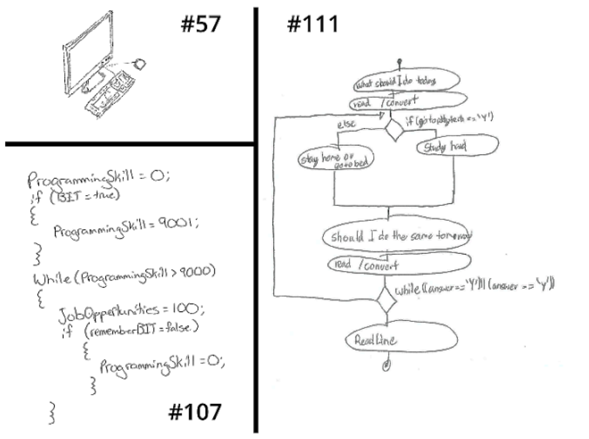
5.4 Aspirations

Some of the drawings (19%, n=77) depicted future-thinking imagery, such as goals, hopes or ambitions, or motivations for learning programming. Ambitions or motivations included: wanting to 'create something' (8%, n=32); money or fame (7%, n=26); job or career (3%, n=13); being able to contribute to the world or future society (3%, n=10); and a degree or grades/marks (2%, n=7). Examples of aspirations are shown in Figure 4.

5.5 Affect

As well as objects and actors, the majority of student drawings were also interpreted as having an emotional or affective component (67%, n=265). This was typically exhibited through symbols such as smiling or frowning faces, or through scenarios such as harm or misfortune befalling an actor; we did not code images where no clear emotions were indicated, or where the emotional 'reading' of the image was disputed by the three coders. Some drawings were coded with multiple emotions, particularly ones divided into different sections (i.e. in a 'comic strip' style), or where textual hints had been added, thus clearly explaining the different emotional states represented.

Positive emotions (such as 'happy/generally positive', 'enjoyable/fun' or 'success/achievement') were found in 38% (n=149) of student drawings. However, there was a noticeable difference between the P1 and P2 datasets—46% (n=95) of P1 drawings showed positive affect, compared with only 28% (n=54) of P2 drawings. We theorise possible explanations of this discrepancy in the Discussion section.



```
ProgrammingSkill = 0;
if (BIT = true)
{
    ProgrammingSkill = 9001;
}
While (ProgrammingSkill > 9000)
{
    JobOpperkunities = 100;
    if (rememberBIT = false)
    {
        ProgrammingSkill = 0;
    }
}
```

Figure 1. Examples of student drawings depicting programming artefacts. Drawing #57 shows a monitor with peripherals (mouse and keyboard); #107 shows a block of pseudocode; and #111 shows a logic/flow diagram.

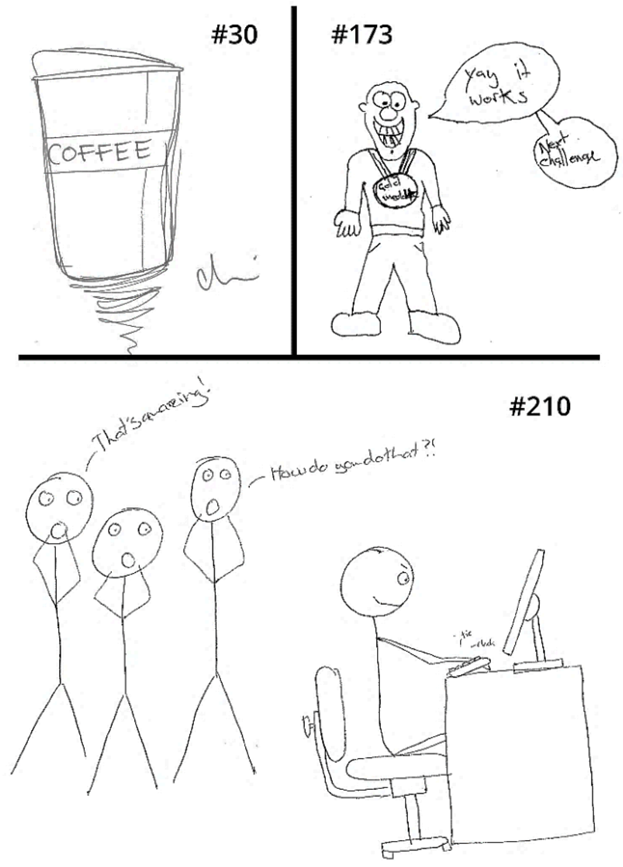


Figure 2. Examples of student drawings depicting different actor configurations. Drawing #30 shows no actor present; #173 shows one actor present; and #210 shows more than one actor present.

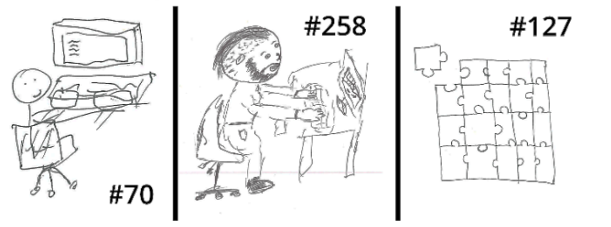


Figure 3. Examples of student drawings showing activities related to programming. Drawings #70 and #258 depict programming as an activity (hands on keyboard); #127 shows a drawing interpreted as 'problem solving'.

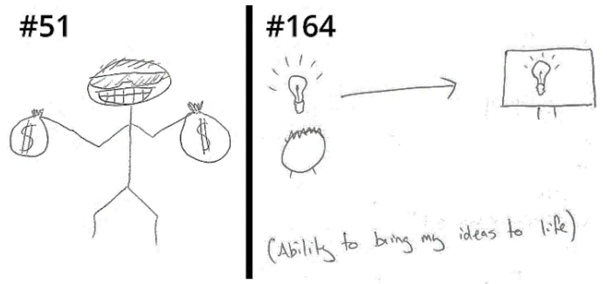


Figure 4. Examples of student drawings showing programming aspirations. Drawing #51 depicts being motivated by 'money or fame'; #164 shows being motivated by the ability to 'create something'.

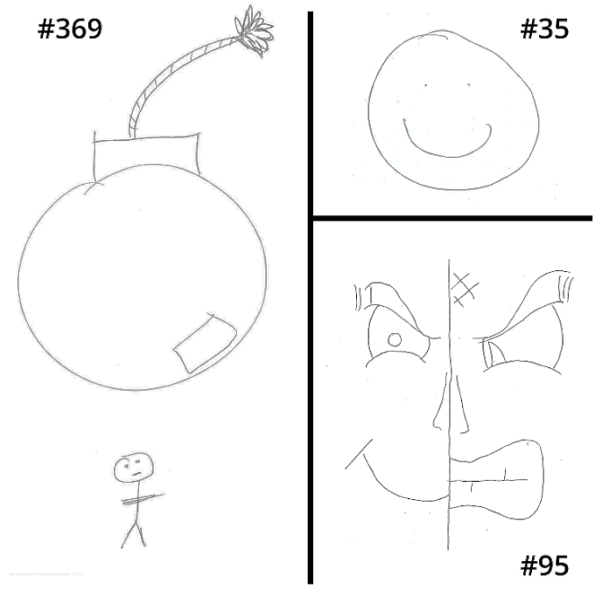


Figure 5. Examples of student drawings showing affect/emotion. Drawing #369 is coded as 'despair'; #35 is coded as 'happy/generally positive'; and #95 is coded as 'mixed emotion'.

Negative emotions (such as 'frustration', 'confusion', 'anger/rage' or 'despair') were found in 28% (n=109) of student drawings. Again, there was a difference between P1 and P2 data, although not as severe as the positive emotion data—23% (n=48) of P1 drawings showed negative affect, compared with 32% (n=61) of P2 drawings.

A comparatively small proportion of drawings (15%, n=60) were coded as 'mixed emotions'—these drawings tended to show conflicting emotions within a single image (i.e. positive and negative emotions in one symbol, such as a halved face with both a smile and frown). Drawings that exhibited some sort of progression over time (either linear or cyclic) between positive and negative emotions were also marked as 'mixed'. Examples of drawings showing affect/emotion are exhibited in Figure 5.

6 DISCUSSION

"Good art is not what it looks like, but what it does to us."
— Roy Adzak

Asking students to describe what programming means to them provides some insight into how students visualise both the act of programming and their role in relation to programming. The use of drawing methodology certainly elicited emotional responses from many students. While it is difficult to draw any definitive conclusions from our interpretations of the student drawings, the data do provide some insights into the student 'way of thinking', and raise some interesting questions for further discussion. In this section, we present observations that we found particularly interesting or surprising, and reflect on how these insights relate to our teaching practice.

Overall, the student drawings depicted programming from a very narrow perspective. Notably, programming as an activity was almost always depicted in conjunction with classroom imagery or references to working on specific in-class assignments; drawings of programming outside of the confines of the classroom were rare. While it may not be surprising that novice programmers' conceptions of programming are primarily tied to their classroom experiences, it does raise questions about student professional identity formation—that is, developing from 'programming students' to 'programmers'. Specifically, when do novice programmers develop a sense of professional identity, and how can introductory programming courses better encourage this development?

A number of studies identify the central role of higher education institutions to foster and facilitate students' emerging professional identities (for a comprehensive review of the literature, see [30]). Reid, Dahlgren, Petocz and Dahlgren [31:738-739] note the key role that strong professional identity formation plays in student learning, stating "professional expectations and values influence the ways that students engage with their learning" and "students find relevance for learning through the obvious applicability of their knowledge".

One specific example from the drawings that suggests student professional identity might not be developing in these introductory courses was a distinct lack of multiple actors across all drawings (Figure 6).

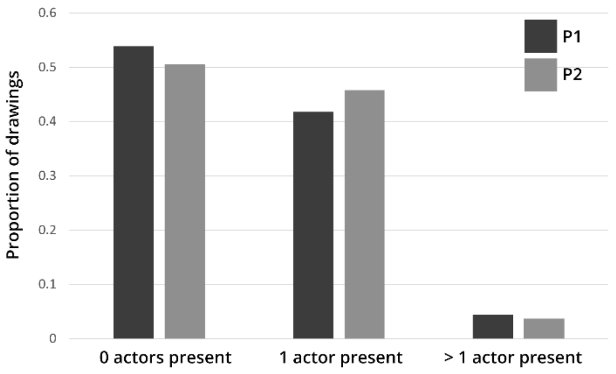


Figure 6. The distribution of drawings with no actors present, one actor present and more than one actor present between P1 and P2.

Collaboration and teamwork are core components of successful software development teams [32]; fostering these attributes as part of a novice programmer's emerging professional identity is an important goal for higher education institutions. Collaborative strategies such as 'pair programming' are widely acknowledged as effective for introductory computing courses [33] In our specific context, researchers B and C both explicitly teach pair programming as a core component of P1 and P2, and have found it to be a particularly beneficial practice for their first-year programmers [34].

The lack of multiple actors in the student drawings, however, suggests that despite continuous exposure to pair programming, social aspects of programming may not be regarded as particularly important by novice programmers. Observations by all three researchers of students in their second year support the idea that pair or social programming is not yet embedded in everyday student practice at this novice stage. As students progress through the later stages of the degree programme (i.e. third year), these collaborative traits tend to be better developed, and this is likely due to the students' involvement in larger-scale, group software engineering projects.

A possible explanation is that pair programming in first year—while demonstrating academic benefit and being viewed positively by students—is being perceived by novices as a teaching and learning strategy, not as a professional skill or part of everyday programming practice. As students experience more 'real-life' programming situations in later years, and begin to develop their identities as 'programmers' rather than 'students', these collaborative practices transition from the academic realm to the professional realm.

Tied to this notion of professional identity formation, relatively few drawings depicted programming as a part of the students' future (i.e. career). When 'future-thinking' imagery was seen, drawings tended to show *unrealistic* conceptions of programming careers, such as programmers surrounded by big bags of money. Jenkins [35:56] found that a 'lucrative career' was a powerful motivator for programming students to undertake computing degrees, but that they may also "[approach] programming from an ill-informed position" regarding the actual skills necessary to get programming jobs.

This misinformed/unrealistic career conception is to be expected in novice programmers—as Leventhal and Chilson [36] found, prolonged exposure to computer science can influence student career expectations, and, particularly, decrease the priority of extrinsic job features, such as monetary compensation. However, the question remains as to how we as computer science educators can better help students visualise themselves in programming careers, and develop healthy expectations of what those careers are likely to be.

The other principal observation we made about the student drawings was the high level of affect present—two-thirds of all student drawings showed clear emotional reaction to programming. As discussed previously, learning to program can be an emotional experience, and this can impact on student performance [8, 9].

There are a number of questions here around how much emotional load is healthy for students to experience in introductory programming, and how can we, as educators, tap into that emotional data to make decisions about our courses?

We highlight an example from the student data related to these questions. There was a noticeable difference between the proportion of positive and negative emotions depicted in the P1 and P2 drawings—overall, the P1 drawings exhibited more positive affect imagery and less negative affect imagery (Figure 7).

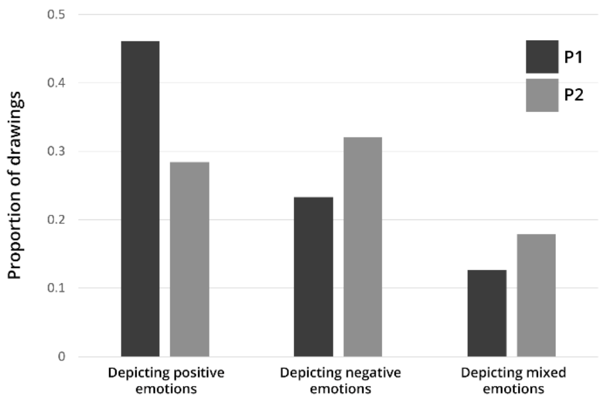


Figure 7. The distribution of drawings showing positive, negative and mixed emotions between P1 and P2.

However, this data is at odds with other measures of student experience in these classes, such as the results of student satisfaction surveys. For instance, the results of the 2016 student satisfaction surveys show the same levels of positive feedback between P1 to P2 (80% and 81% overall satisfaction respectively). Our drawing data, though, would suggest that the student experience of these classes is more nuanced than the relatively blunt satisfaction metric indicates.

Recent research into student 'happiness' supports this conjecture—Dean and Gibbs [37:16] found that indicators of student 'happiness' and student 'satisfaction' tend to be different. They found that 'satisfied' students "seemed to be more concerned with external loci, that is, on how things done to and for them were delivered, rather than in their engagement with the process."

Researchers B and C theorise that this perceived decrease in positive affect as students move from P1 to P2 is likely due to the conceptual shift required between the two subjects; specifically, from:

- programming fundamentals to abstract OO concepts;
- console-based development to using an IDE and GUI (Graphical User Interface); and
- discrete tasks aimed at skill development, to more open-ended applications of skills.

For example, the difficulties in getting novice programmers to grasp OO concepts is a well-documented challenge for CS educators [38, 39]; anecdotally, researchers B and C share this view that students find programming in P2 more confusing than P1.

However, a level of challenge (i.e. confusion of frustration) can be beneficial to the learning process. This concept is widely espoused in educational literature: Wass and Golding [40] suggest pitching teaching at the extreme limits of a student's zone of proximal development (ZPD, or tasks too hard to do independently, but achievable with assistance); Bjork and Bjork [41] talk about 'desirable difficulties' or challenges that can help trigger cognitive process and enhance learning; and D'Mello and Graesser [42] argue that the 'cognitive disequilibrium' experienced during difficult tasks leads to deeper learning.

However, as with many theories of learning, transforming these concepts into practical advice for teachers is less straightforward. For example, Wass and Golding [40] acknowledge that determining the actual boundaries of a student's ZPD is difficult; similarly, D'Mello and Graesser [42] caution that too much disequilibrium can have the opposite effect, causing students to give up or become disengaged, and tasks should be tailored to individual learners.

We suggest that drawing methodology might offer a useful approach for teachers to tease out student affective states, and hence infer the level of challenge currently experienced by students. When viewed as group data, our drawings give an overall 'emotional snapshot' of the course—we would expect to see a balance of positive and negative emotions across the entire cohort of students. Too much positive emotion could indicate the course is not challenging enough (and thus not conducive to deep learning), while too much negative emotion could indicate the course is too challenging (and thus lead to student disengagement).

Obviously, this type of exploratory and interpretive data is problematic: the analysis process is time-consuming; the process is heavily dependent on context, and therefore not easily transferrable to other teaching scenarios; and it is difficult to draw definitive conclusions from the data. However, all three researchers found the student drawing data offered a far richer picture of student in-class experiences than other traditional measures, and this in turn sparked extensive discussions and reflection around the content and delivery of our introductory programming courses. This sort of deeply personal data gave us a window into our students' experiences, and helped us remember that our students are more than pass-rates and enrolment dollars—they are complex and emotional people.

7 FUTURE RESEARCH

Throughout our research, we noted several potential directions for future research.

First, we are curious to know whether repeating this exercise with second and third-year students would reveal any different patterns than the first-year cohorts. Specifically, we are interested in seeing whether more indicators of student professional identity formation are present in drawings from students further along in the degree.

Second, similar to above, we would be interested in getting graduates and industry professionals to complete the drawing activity to see if this cohort produces any significantly different drawings. Comparing student drawings to an industry 'exemplar' set of drawings could reveal further insights into student experiences.

Third, in this study we did not explicitly correlate student drawings with performance; this was for two main reasons: (1) as reported, we were more interested in looking at patterns across the data as a group; and (2) a preliminary look at a sampling of the drawings did not show any obvious correlations to student performance (i.e., the drawings were attached to their final exam papers, and there did not seem to be any obvious patterns relating to the exam marks). However, a more thorough exploration of the five themes (artefacts, actors, activities, aspirations and affect) in conjunction with student performance measures should be carried out in future.

Fourth, to augment our methodology, we could repeat the activity with students and incorporate a discussion element to assist with our analysis. This might involve sitting down with students and letting them explain their drawings in a semi-structured interview fashion, or providing students with our analyses and comparing and contrasting our interpretations with those from a student perspective.

Fifth, we acknowledge that the timing of the exercise—that is, at the end of a final exam—may be confounding the data, particularly around our interpretations of student emotions. We came up with two possible issues that warrant further investigation to substantiate: (1) the drawings may be a cathartic outlet for students following a stressful event (e.g. the exam), and emotions such as frustration, rage, relief, or joy may pertain specifically to the exam, rather than the overall course itself; and (2) particularly positive emotions about the course could simply be placatory toward the lecturer about to mark the exam paper. This brings up questions around the possible teacher-student power relations embedded in the drawings [22].

Finally, there are several demographic comparisons that we could focus on with future studies: for example, traditionally computer science has a challenge with attracting and retaining female students, and it could be worthwhile seeing if male and female student drawings produce different conceptions of programming (of course, with so few female students, getting enough drawings to constitute a viable dataset could be a challenge). Other demographics that would be of interest to our context would be whether we see differences with Māori and Pacific Island students, or between school-leavers and adult students.

8 CONCLUSIONS

As with any good art, our student drawings have provoked questions and reflection. As a window into the student experience of learning programming, the student drawings reveal a rich dataset of conceptions, perspectives, aspirations and emotions. The observed patterns noted in the drawings raised questions particularly around the formation of students' professional programming identities, and the role of affect in learning to program. As educators, we are constantly looking for new ways to improve our teaching practice and the learning experiences of our students. In this study, drawing methodology proved to be a useful means of tapping into the affective states of our first-year students and gaining insight into their learning experiences.

ACKNOWLEDGMENTS

Thanks to all the students that took the time to produce a drawing.

REFERENCES

[1] D. Knuth. 1974. Computer programming as an art. *Communications of the ACM, v.17 n.12*, pp.667-673, Dec 1974.

[2] E. Lahtinen, K. Ala-Mutka, and H-M. Järvinen. 2005. A study of the difficulties of novice programmers. *Acm Sigcse Bulletin, vol. 37, no. 3*, pp.14-18. ACM, 2005.

[3] A. Robins, J. Rountree, and N. Rountree. 2003. Learning and teaching programming: a review. *Computer Science Education.* 13, 2, pp.137-172.

[4] A. Berglund, and R. Lister. 2010. Introductory programming and the didactic triangle. In *Proceedings of the Twelfth Australasian Conference on Computing Education-Volume 103* (pp. 35-44). Australian Computer Society, Inc.

[5] M. Butler, and M. Morgan. 2007. Learning challenges faced by novice programming students studying high level and low feedback concepts. In *Proceedings of the 24th ascilite Conference* (pp. 2-5).

[6] A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson. 2007. A survey of literature on the teaching of introductory programming. *ACM SIGCSE Bulletin, 39*(4), pp.204-223.

[7] S. Fincher, A. Robins, B. Baker, I. Box, Q. Cutts, M. de Raadt, P. Haden, J. Hamer, M. Hamilton, R. Lister, and M. Petre. 2006. Predictors of success in a first programming course. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52* (pp. 189-196). Australian Computer Society, Inc., Jan 2006.

[8] P. Kinnunen, and B. Simon. 2012. My program is ok–am I? Computing freshmen's experiences of doing programming assignments. *Computer Science Education, 22*(1), pp.1-28.

[9] J. Gasson, D. Parsons, K. Wood, and P. Haden. In review. Student affect in CS1: Insights from an easy data collection tool. *Koli Calling*, November 16–19, 2017, Koli, Finland.

[10] C. Edmondson. 2008. Teaching tales: some student perceptions of computing education. *ACM SIGCSE Bulletin, 40*(4), pp.103-106.

[11] L.C. Kaczmarczyk, E.R. Petrick, J.P. East, and G.L. Herman. 2010. Identifying student misconceptions of programming. In *Proceedings of the 41st ACM technical symposium on Computer science education* (pp. 107-111). ACM. Mar 2010.

[12] J.C. Liang, Y.C. Su, and C.C. Tsai. 2015. The assessment of Taiwanese college students' conceptions of and approaches to learning computer

science and their relationships. *The Asia-Pacific Education Researcher*, *24*(4), pp.557-567.

[13] I. Stamouli, and M. Huggard. 2006. Object oriented programming and program correctness: the students' perspective. In *Proceedings of the second international workshop on Computing education research* (pp. 109-118). ACM. Sep 2006.

[14] D. Krpan, M. Rosić, and S. Mladenović. 2014. Teaching basic programming skills to undergraduate students. In *Contemporary issues in economy and technology.* Jan 2014.

[15] A. Eckerdal, M. Thuné, and A. Berglund. 2005. What does it take to learn 'programming thinking'?. In *Proceedings of the first international workshop on Computing education research* (pp. 135-142). ACM. Oct 2005.

[16] S. Fincher, J. Tenenberg, and A. Robins. 2011. Research design: necessary bricolage. In *Proceedings of the seventh international workshop on Computing education research* (pp. 27-32). ACM. Aug 2011.

[17] T. Hübscher-Younger, and N.H. Narayanan. 2003. Dancing hamsters and marble statues: characterizing student visualizations of algorithms. In *Proceedings of the 2003 ACM symposium on Software visualization* (pp. 95-104). ACM. Jun 2003.

[18] C. Golomb. 2003. *The child's creation of a pictorial world.* Psychology Press.

[19] R.P. Jolley. 2010. *Children and pictures: Drawing and understanding.* John Wiley & Sons.

[20] M.R. Lea, and B.V. Street. 1998. Student writing in higher education: an academic literacies approach. *Studies in higher education 23*(2), pp. 157-172.

[21] K. Hyland. 2008. As can be seen: lexical bundles and disciplinary variation. *English for specific purposes, 27*(1), pp. 4-21.

[22] M. Guillemin. 2004. Understanding illness: Using drawings as a research method. *Qualitative health research*, *14*(2), pp.272-289.

[23] N. Selwyn, S. Boraschi, and S.M. Özkula. 2009. Drawing digital pictures: An investigation of primary pupils' representations of ICT and schools. *British Educational Research Journal*, *35*(6), pp.909-928.

[24] W.M. Hsieh, and C.C. Tsai. 2016. Learning illustrated: An exploratory cross-sectional drawing analysis of students' conceptions of learning. *The Journal of Educational Research*, pp.1-12.

[25] W.M. Hsieh, and C.C. Tsai. 2017. Exploring students' conceptions of science learning via drawing: a cross-sectional analysis. *International Journal of Science Education*, pp.1-25.

[26] K.N. Sim. 2016. *An investigation into the way PhD students utilise ICT to support their doctoral research process* (Doctoral dissertation, University of Otago).

[27] S. Köse. 2008. Diagnosing student misconceptions: Using drawings as a research method. *World Applied Sciences Journal*, *3*(2), pp.283-293.

[28] F. Cornish, A. Gillespie, and T. Zittoun. 2013. *Collaborative analysis of qualitative data.* Sage Publications Limited.

[29] P.A. Nielsen. 2016. Towards a Design Theory for Collaborative Qualitative Data Analysis. *Practice-based Design and Innovation of Digital Artifacts.*

[30] F. Trede, R. Macklin, and D. Bridges. 2012. Professional identity development: a review of the higher education literature. *Studies in Higher Education*, *37*(3), pp.365-384.

[31] A. Reid, L.O. Dahlgren, P. Petocz, and M.A. Dahlgren. 2008. Identity and engagement for professional formation. *Studies in Higher Education*, *33*(6), pp.729-742.

[32] Y. Lindsjørn, D.I. Sjøberg, T. Dingsøyr, G.R. Bergersen, and T. Dybå. 2016. Teamwork quality and project success in software development: A survey of agile development teams. *Journal of Systems and Software*, *122*, pp.274-286.

[33] C. McDowell, L. Werner, H. Bullock, and J. Fernald. 2002. The effects of pair-programming on performance in an introductory programming course. *ACM SIGCSE Bulletin*, *34*(1), pp.38-42.

[34] K. Wood, D. Parsons, J. Gasson, and P. Haden. 2013. It's never too early: pair programming in CS1. In *Proceedings of the Fifteenth Australasian Computing Education Conference-Volume 136* (pp. 13-21). Australian Computer Society, Inc. Jan 2013.

[35] T. Jenkins. 2001. The motivation of students of programming. In *ACM SIGCSE Bulletin* (Vol. 33, No. 3, pp. 53-56). ACM. Jun 2001.

[36] L.M. Leventhal, and D.W. Chilson. 1989. Beyond Just a Job: Expectations of Computer Science Students. *Computer Science Education*, *1*(2), pp.129-143.

[37] A. Dean, and P. Gibbs. 2015. Student satisfaction or happiness? A preliminary rethink of what is important in the student experience. *Quality Assurance in Education*, *23*(1), pp.5-19.

[38] M. Kölling. 1999. The problem of teaching object-oriented programming. *Journal of Object Oriented Programming*, *11*(8), pp.8-15.

[39] K. Sanders, J. Boustedt, A. Eckerdal, R. McCartney, J.E. Moström, L. Thomas, and C. Zander. 2008. Student understanding of object-oriented programming as expressed in concept maps. *ACM SIGCSE Bulletin*, *40*(1), pp.332-336.

[40] R. Wass, and C. Golding. 2014. Sharpening a tool for teaching: the zone of proximal development. *Teaching in Higher Education*, *19*(6), pp.671-684.

[41] E.L. Bjork, and R.A. Bjork. 2011. Making things hard on yourself, but in a good way: Creating desirable difficulties to enhance learning. *Psychology and the real world: Essays illustrating fundamental contributions to society*, pp.56-64.

[42] S. D'Mello, and A. Graesser. 2012. Dynamics of affective states during complex learning. *Learning and Instruction*, *22*(2), pp.145-157.