A SURVEY OF PROGRAMMERS' PRACTICES FOR HANDLING COMPLEXITY IN CREATIVE CODING

OLLIE BOWN

Design Lab, University of Sydney, Darlington, NSW, 2006 Australia

oliver.bown@sydney.edu.au

ROB SAUNDERS

Design Lab, University of Sydney, Darlington, NSW, 2006 Australia

rob.saunders@sydney.edu.au

MARTIN TOMITSCH

Design Lab, University of Sydney, Darlington, NSW, 2006 Australia

martin.tomitsch@sydney.edu.au

Keywords: Creative Coding, Generative Design, Complexity

Creative coding has goals, methods and tools that distinguish it from other forms of programming practice. A number of creative coding practitioners regularly engage with complex systems in creative applications, such as neural networks, genetic evolution, ecosystems and reaction-diffusion equations. Such systems are conceptually opaque and hard to manipulate, but we know little about the ways in which creative coders handle them. In this paper we present findings from a survey of creative coders regarding how they deal with complexity in their work. We discuss four issues of interest: time-demands of specific activities; sources of knowledge and approaches to problem solving; approaches to exploration; and information-seeking versus intuitive styles of working. Our results provide an initial characterisation of creative coding strategies and suggest how tools could be better adapted to their needs.

1. INTRODUCTION

Creative coding or creative computing has begun to emerge as a field that can be distinguished from other types of computer programming, based on the nature of the practice, the tools used and the methods employed in its production (McLean and Wiggins, 2010; Turkle and Papert, 1990; Mitchell and Bown, 2013). Researchers are beginning to focus on these areas in order to understand how creative coders can get the most out of what they do, either through new tools or new forms of practice.

As an example, the idea of merging code and GUI elements has been explored in many contexts, leading to a more flexible integration between writing code and exploring its parameter-space: in visual programming languages, such as MaxMSP and Pure Data, GUI elements sit within the visual programming paradigm; in hybrids, such as Field, GUI elements can be inserted into blocks of code; in web-based environments, such as Khan's Code Academy, variables such as numbers and colours in the code can be interacted with to change their values; and in add-ons to regular IDEs, such as the Processing environment optional add-on, Tweak Mode, the same effect is achieved working directly on the Java code. The utility of such tools are easily inferred from their popularity, but more detailed studies of what transformative effects they have on creative coding practice are also required (see Mitchell and Bown, 2013 for some recent analysis).

2. BACKGROUND

In previous work examining the potential to apply complex dynamics from models of nature to creative goals (McCormack and Bown 2009, Bown and McCormack 2010), a key observation was that, whilst rich biological models – such as those of ecosystems evolution – had creative value, the biggest obstacle to their successful use was the difficulty of working with their unwieldy complexity. Our autobiographical accounts of programming such systems showed that we often didn't have good enough information about what the system was doing to be able to make clear informed creative choices about what steps to take next.

All software programmers must take steps to discover things about their running programs that they cannot tell from the code alone. This often involves gleaning information that is additional to the output itself. Thus

a programmer animating a flock may already have the flock to look at on their screen, but they may add print statements to reveal other data – in the form of text – to the screen or to a console output, view variables in a debugger, draw additional layers on the animation, such as velocity vectors or the flock's centroid, or use other data visualisation methods such as histograms or scatterplots. Some tools exist for these purposes, such as debuggers and convenient libraries for visualising data. Other forms of data discovery need to be added by the programmer. For example, typically the programmer would draw visual layers such as velocity vectors themselves, but they could equally use a third-party physics library that provides visualisation layers for them.

As well as gleaning information about their programs, creative coders regularly iterate through a number of designs in search of specific outcomes. An additional form of information acquisition therefore comes in the form of partial automation of their search process. The flock programmer may have a number of real-valued variables that they wish to explore in order to find the system's best settings. In that case they may explore this parameter space using one of the interactive methods described above. Alternatively, they could perform a *parameter sweep*, in which a large number of parameter values are input in order to produce many different possible animations (perhaps generated overnight while the programmer sleeps), which can be quickly examined. A more advanced approach would be to conduct some kind of automated search for preferred parameters, but this requires the flock programmer to define in the program what is being sought. Whilst all of these approaches are feasible, they require increasing levels of input to achieve desirable results, sometimes involving significant design challenges in their own right.

Finally, information about the workings of a program comes not only from isolated inspection of the program itself but by learning from other people: the flock programmer has likely discovered the classical flocking algorithms through books or lectures or discussions with colleagues, and may also find out the precise implementation and specific parameter values that way. If a programmer's code produces unexpected results, they may be able to understand why through discussion and comparison with others' results. This form of knowledge acquisi-

tion is evidently ubiquitous throughout programming practice: we would not even know of the existence and applications of cellular automata, reaction-diffusion equations or neural networks without learning about them from others. It is rare that someone makes an original discovery as significant as one of these canonical forms.

In this paper we present the initial findings from a study that sets out to understand how practitioners approach knowledge acquisition and manage their understanding of their code when interacting with software objects. Our focus is on the more complex end of topics that creative coders work with: concepts from computer science such as genetic algorithms, reaction-diffusion systems and self-organising behaviour. For the purpose of this discussion we loosely refer to this set of elements as 'complex systems' (although note that 'complex systems science' refers to an overlapping but distinct set of systems).

Complex systems have in common that certain aspects of their behaviour are opaque to the user. Although this could be said of computer programs in general, complex systems push the cognitive capacities of the programmer to the point where it is not viable to maintain a mental model of what the system is doing at all levels of scale, whereas for many acts of programming the programmer is able to maintain a clear understanding of the macroscopic effects of specific programmed behaviours. With complex systems, the system changes and interacts in ways that are not clearly predictable or transparent. For example, in artificial evolution the structure of a computational component, the genotype, is modified by a process of selection and random mutation, and its outcome may not be easily predicted from the design of the process. In this way, surprising designs come about, such as those of Sims' famous evolved virtual creatures (Sims, 1994). Complex systems are, as their name implies, hard to work with.

3 . STUDY

For all of the strategies mentioned above, there is little information on the extent to which creative coders use them, how they enhance creative search, and where and when they are most effective. Through this study we set out to begin to understand what aspects of the programming process support or hinder creative development using these systems.

The study was run as an online survey.[1] Participants were asked to respond to the survey only if they worked with complex systems, a list of examples of which were provided at the start of the survey. Participants were asked a number of background questions regarding their training, experience, the tools they use, and the application areas they work in. They were then presented with a number of statements with Likert-scale response options (strongly disagree, disagree, neutral, agree, strongly agree) on a range of issues from their approach to debugging, to their experience of surprise. All questions had the option of adding additional comments.

As a provisional look at practice, the questions covered a broad range of topics and were designed to give clues as to where to look in further more detailed research. Our aim was to use the current survey results to identify compelling areas of further study.

4. RESULTS

Participants were contacted via special interest mailing lists catering for communities working creatively with code (with which we include visual programming environments such as MaxMSP), such as the Processing forum and the Computer Arts Society mailing list. 110 respondents started the survey and 39 completed it entirely. All questions were optional and no responses were discarded. The average age of respondents was 41.1 years. Of those who chose to respond, 53 were male and 7 were female, confirming a strong and widely recognised gender imbalance in creative electronic arts.

We divide results into four distinct areas: time-demands of specific activities; sources of knowledge and approaches to problem-solving; approaches to exploration; and information-seeking versus intuitive styles of working. We present results in each of these areas interspersed with some discussion.

4.1. TIME DEMANDS OF SPECIFIC ACTIVITIES

One of the sets of questions with the most unanimity concerned the time-burden of different parts of the programming process. Most participants (40.5% SA, 40.5% A)[2] reported that along with the coding of the system itself, “a significant proportion of time was spent exploring spaces of possibilities (e.g., parameter spaces) of the system”. Likewise, a moderate number of participants

1 The full survey, with fully anonymised numerical only results, can be found at: *http://2014.xcoax.org/files/070.zip*

2 For reporting statistics we give the percentage of respondents stating Strongly Agree (SA), Agree (A), Neutral (N), Disagree (D) and Strongly Disagree (SD).

(13% SA, 25% A 22%N) reported that it was hard to find the system's 'sweet-spots', whilst the vast majority (60% SA, 15% A) reported that they had experiences of finding surprising behaviour. A moderate number of participants (19% SA, 17% A, 19% N) also reported spending significant time on writing unit tests. Two participant's comments suggest that the central challenge of their work lay outside of the domain of programming *per se*:

> *"The system was 'easy' to implement in the sense that there were no new technical breakthroughs required in order to make it work. However, figuring out how to make it work aesthetically was complex and time consuming." (Respondent 50)*

> *"I definitely understood the low-level behaviors, but was continuously amazed by higher-level emergent behaviors." (Respondent 56)*

This suggests that it may make sense to distinguish aesthetic search and/or design from the act of programming in creative coding. But another participant's remarks related to the difficulty of setting debugging issues apart from design in this context:

> *"I did not understand what was going on due to complex bugs in my code" (Respondent 43)*

Finally, most respondents reported satisfaction with the high-level libraries available to them (49% SA, 25% A), i.e., they did not feel that a lack of high-level libraries was a hindrance to progress.

We may ask then whether existing tools are catering sufficiently for the time-demands of creative programmers, given that significant time is spent in an exploratory state. For example, it may be that tools that allow breaking out of a programming mode of activity and into a design mode of activity would be useful. Further study into this area, could involve prototyping and user-testing such a tool to understand its efficacy.

4.2. SOURCES OF KNOWLEDGE AND APPROACHES TO PROBLEM SOLVING

We asked respondents about the different ways in which they inspected their own programs, considering graphs and statistics, abstract visualisations and indirect quanti-

tative measures such as entropy. The dominant approach identified in responses was abstract visualisation (49% SA, 15% A) (the example we give above of abstract visualisation is the visualisation of mechanical properties such as velocity vectors, although we did not offer specific examples to respondents), which the majority of respondents used, though all methods received above neutral average responses. We also looked at how people used personal communication and other knowledge resources to better understand their systems. We found a greater tendency to solve problems alone (47% SA, 39%A), e.g., using deduction or trial and error, rather than seeking help directly (34% SD, 16% D) or on forums (56% SD, 21% D).

We do not have data on regular programming practice to compare this to. Logically, individual problem solving must be more common than consulting forums, since it is necessary to try and solve a problem before asking for help. However, the lack of use of forums may be related to the idiosyncrasy of systems, and the loosely-defined nature of problems in this context, that would make it harder for others to get involved. As above, there is also a distinction to be made, as well as an overlap, between programming problems and design problems. Respondents generally agreed that the idiosyncrasy of their systems limited the value in seeking help. A study of creative coding forums could be used to reveal more information about the level at which individual design issues are raised.

4.3. APPROACHES EXPLORATION

Artistic creativity is often described as exploratory, and a number of personal accounts of creativity in the digital arts known to the authors emphasise search as a core process. One comment from a respondent expresses a common scenario for creative coders familiar to the authors:

> *"I was hoping to get sequences that were on the 'borderline' of randomness and predictability. In fact, the series almost always ended up too random." (Respondent 7)*

Anecdotally, we have noticed in our own work that it is common to have expectations of specific phenomena that do not materialise easily in practice. It may then be

common to have a mismatch between expectation and outcome. It would be reasonable to guess that the expectation was too great. But it could also be that in such cases the programmer is actually close to achieving their goals but without the required tools or methods to ultimately find the desired solutions.

Other approaches are more pragmatic in that there is no search for an ultimate configuration, only for good candidates, which can be integrated simply by switching between states:

> *"I actually ended up using several different permutations of the flocking system in the one work." (Respondent 43)*

A large proportion of participants (66% SA, 24% A) reported that one motivation for using their chosen complex system was that it was a good generator of variation. The sentiment may therefore be common to a wide-range of creative computing objectives, typical in the practice of 'generative art'. Techniques for rapidly generating variation, and doing so for a wide range of aesthetic objects, may therefore be the most *currently* useful tools in the creative programmer's toolkit. Finally, respondents tended to agree that batch processing in their search for solutions was within their capacity (59% SA, 19% A), and a moderate number reported a willingness to use batch processing (43% SA, with an even distribution across the rest of the scale).

4.4. TYPES OF CREATIVE CODER

Based on our own experience we hypothesised that a distinction may exist between two prototype styles: an information-seeking approach which emphasised the need to manage the complexity of the system through analysis, valued tools that supported this, and sought additional knowledge about systems; and an explorative approach which emphasised an intuitive iterative creative cycle with a focus on visualisation. This latter type may correspond to the archetype of the creative, 'bricoleur' programmer (McLean and Wiggins 2010), and closer resemble the artistic practice of a painter, sculptor or composer, who is engaged in a *tight coupling* of action and response with their artistic material. This is suggested in a comment by one such practitioner:

> *"I regard the creation of algorithms as similar to a traditional compositional activity… understanding can be intuitive and 'musical'. " (Respondent 96)*

The former type resembles a more traditional view of an engineering approach, but could be more exploratory than software development in non-creative domains, distinguished by an openness to the final outcome.

The same participant also stated:

> *"I don't always know exactly how the system will work when finished; and I prefer not to aim for a preconceived goal." (Respondent 96)*

This may be distinctive of an exploratory approach, but we suspect that it is not actually the distinguishing feature between approaches – both approaches could accommodate an open-mindedness towards the final outcome.

In fact, the numerical results did now reveal clear-cut distinctions between types of practice, but they did show a wide range of responses, as well as significant correlations between sets of responses to those questions that asked whether better tools were needed to enhance creative search. According to these correlations, participants' level of contentment with their development environment was not specific to any particular improvements, but rather generic. However, there was no correlation, positive or negative, between how advanced people's use of analysis tools was and their recognition of a need for better analysis tools or methods.

These results suggest that there is no neat distinction between information-seeking and intuitive approaches, but do support the idea that some programmers are content with methods for discovering information whilst others desire improved support tools. This may include different information-seeking needs. We therefore suggest that there is good reason to seek innovative and diverse ways to support creative coders with information about the systems they are coding. We also propose that further research should continue to try to identify distinct strategies for handling complexity, given our wide-range of responses, and to further understand how practitioners define goals and go about arriving at outcomes with respect to those goals.

5. CONCLUSION

This study takes a first detailed look at the way creative coders manage complexity in their practice. Our results provide pointers for how to think about categorising and understanding creative coding practice, indicating a range of approaches to handling complexity. We have summarised four areas where we feel further research could be carried out into how creative coders work and what tool designs would support them. We have briefly discussed further questions, and possible implications for future creative coding tools.

The following conjectures are not conclusive given the current results but are at least supported by them and worthy of further study: exploratory search is a major component in creative coding which could be better supported by allowing it to take place in a non-programming design phase; additional and varied forms of feedback may provide one way to enhance the search process, and a range of distinct information-seeking strategies should be considered; the generation of variation is currently a key motivator for creating with code, and is perhaps a greater focus for creative practitioners than discovering sweet-spot states, possibly because the latter is harder to achieve.

REFERENCES

Bown, Oliver and McCormack, Jon. "Taming Nature: Tapping the creative potential of ecosystem models in the arts", Digital Creativity, 21(4), pp. 215–231, 2010.

McCormack, Jon. "Facing the future: Evolutionary possibilities for human-machine creativity." The Art of Artificial Evolution. Springer, Berlin, 2008. pp. 417–451.

McCormack, Jon and Bown, Oliver. "Life's what you make: Niche Construction and Evolutionary Art". Springer Lecture Notes in Computer Science, Proceedings of EvoWorkshops, M. Giacobini, *et. al.* (eds), LNCS 5484, pp. 528–537, 2009.

McLean, Alex, and Wiggins, Geraint. "Bricolage programming in the creative arts." 22nd annual psychology of programming interest group, 2010.

Mitchell, Mark, and Bown, Oliver. "Towards a creativity support tool in processing: understanding the needs of creative coders." Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration. ACM, 2013.

Sims, Karl. "Evolving virtual creatures." Proceedings of the 21st annual conference on Computer graphics and interactive techniques. ACM, 1994.

Turkle, Sherry, and Papert, Seymour. "Epistemological pluralism: Styles and voices within the computer culture." Signs 16.1: pp. 128-157. 1990.