Towards a Creativity Support Tool in Processing: Understanding the Needs of Creative Coders

Mark C. Mitchell
Design Lab, University of Sydney
Darlington, NSW, 2006

Oliver Bown
Design Lab, University of Sydney
Darlington, NSW, 2006

ABSTRACT
Creative coding as a paradigm has seen increased interest in recent years. However, detailed studies of the processes and needs of these creative coders are currently lacking. This paper reports on the preliminary findings of a study into the practices of both novice and expert creative coders by analysing their approach to a creative design task in an observational, qualitative study. The findings have been placed into a taxonomy of needs, which can feed into the design of tools that aim to assist creative coders. The paper concludes by discussing the implications the taxonomy of needs has on defining requirements for a creativity support tool in the Processing environment.

Author Keywords
Creative coding; processing; creativity support tool; media arts

ACM Classification Keywords
D.2. Software Engineering: Programming Environments. Human-centered computing: User studies.

INTRODUCTION
Creative coders write software for games, art installations, generative artworks, data visualisations, music and so on. They combine the skills of programming with creative arts and design.

The rise in prominence of creative coding as a practice is apparent in the emergence of conferences ("push.conference," 2013), coding libraries ("Cinder," 2013), and in its use as replacement for traditional computer science education (Greenberg, I. et al., 2012).

Over the last decade, a number of creativity researchers have been looking at the use of technology in the creative process. HCI research has been increasingly utilised in crafting creativity support tools to further engage creative activity. HCI principles for creativity support tools have been identified (Shneiderman et al., 2005), and frameworks have been created, such as GENEX (Shneiderman, 2002).

Research into the processes used by software developers to solve tasks is extensive (Miryung et al., 2004; Rosson and Carroll, 1993) however a detailed account of the practice of creative coders is currently lacking, despite growing interest in the theoretical study of creative coding as a practice, such as (McLean and Wiggins, 2010).

Research into the needs of creative coders creates a set of opportunities for supporting creativity. With this goal in mind, we have conducted a study into the practices of both novice and expert creative coders, analysing their approach to a creative design task in the Processing environment. The inherent needs of creative coders are identified and discussed, and the paper concludes with the implications these findings have for the design of creativity support tools for creative coders.

OzCHI'13, November 25 - 29 2013, Adelaide, Australia

BACKGROUND

The creative coder paradigm
Despite the rise in prominence of creative coding, a working description of what it means to "code creatively" is difficult to obtain. We define creative coding as a discovery-based process consisting of exploration, iteration, and reflection, using code as a primary medium, towards a media artefact designed for an artistic context.

This definition allows creative coding to be viewed through the lens of bricolage programming (exploration and iteration), and Schön's notion of reflection-in-action, where practitioners think on their feet and upon immediate reflection generate new theories, which dictate further actions.

Turkle and Papert (1990) describe the approach of the bricolage programmer as one who constructs theories by arranging and rearranging; negotiating and renegotiating with code. This approach exists between a structured formation of code and a creative trial-and-error process: Reas, co-creator of Processing, describes writing software as "a process of translating fuzzy ideas from one's mind into a strict notational system. Using this notation as an intermediate step, visual and kinetic ideas manifest themselves in computational machines" (Reas, 2003).

Processing
Possibly the most influential coding environment designed specifically for the practice of creative coding, Processing was created with a dual goal of teaching fundamentals of computer programming within a media arts context (Reas and Fry, 2006) and to serve as a software sketchbook for visual designers, with an aim to achieve a balance between clarity and advanced features.

This has led to the success of Processing with the creative coding community, who may be seen as having distinct requirements from professional software developers. Processing's popularity extends to both novices and

experts, with the following features that distinguish it from other professional coding environments:

- The focus on "sketching" programs affords fast prototyping of ideas with minimal setup, due to a simplified syntax and graphics-programming model.
- Example sketches are readily available from the interface, grouped into various artistic and computational topics.
- The Integrated Development Environment (IDE) of Processing is highly simplified by design, to ease project overhead and reduce the learning curve. There is no code completion, live compiling, refactoring support, project management tools, or version control.

Processing's popularity suggests that these constitute important design requirements for a creative coding IDE. However, few formal user studies have been conducted into how these elements affect the creative coding process and what effect alternative designs would have.

A FORMATIVE INVESTIGATION

To analyse the practices of creative coders and better understand their needs to inform the requirements of a creativity support tool, a qualitative study was conducted with self-described creative coders, using the Processing environment. Whilst Java (the language underlying Processing) is an object-oriented programming (OOP) language which emphasises modular design, Processing makes limited use of OOP functionality and de-emphasises modularity in its presentation to the user. Our anecdotal experience is that this creates a more manageable learning environment for beginners, and a development environment geared towards quick sketches, but at the cost of modular design, which requires more effort to implement.

The investigation was designed to study the approach creative coders take in a time-limited creative task. Emphasis of the study was placed on code modularity, OOP practices, and the use of third-party libraries.

Methodology

The study was conducted using direct observation and semi-structured interviews with a contextual methodology. Nine participants were recruited. All participants were required to have prior experience writing a Processing sketch in a creative context.

The study was conducted with one participant at a time, who were observed performing the task and then interviewed. The think-aloud method was not employed, as the think-aloud process risks hindering the thought process of the participant whilst writing code. Instead, a post-task interview was used, so the flow of the participant was not interrupted. The observer noted down interesting participant behaviour, and asked them to elaborate on their actions in the post-task interview.

The Task

Participants had one hour to complete the task. The computer screen of the participant was recorded, along with audio recordings of the post-task interview.

A basic Processing sketch that represented a two-dimensional environment was presented. The participants were asked to engage in a task of creating a virtual creature and adding it to the environment. The participants were asked to give the creature an autonomous movement behaviour, and an interesting visual appearance. Three example creatures were displayed, and the ability to access their source code was shown to the participant.

The participants were encouraged to utilise any resources for the task they wished, including code libraries, code from the Internet, the example creatures shown at the beginning of the task, or previous work. Observations and interview questions were tailored to extract the *why* of the practice, rather than the *what* — for example, if the participant was observed searching for example code, it is important to extract not just *what* they are looking for, but the motivating factors behind their action. This allows deeper insights into how the process can be technologically supported (Hewett et al., 2005).

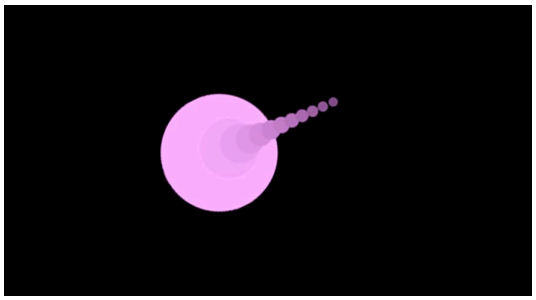

Figure 1. One of the creatures coded by a participant.

RESULTS

Upon conducting a thematic analysis on noted observations during the task and participant responses in the post-task interview, six needs of creative coders have been identified, and placed into a taxonomy of needs.

A Taxonomy of Needs

The nine participants of the study are referred to in the following sections as P1 through to P9:

Make the program state visible

All participants encountered unintended program behaviour. Because of Processing's highly simplified interface (there is limited error messaging and no debugging tool), participants regularly utilised a print statement to see data in the program console, extracting variable values at runtime, to find the cause of unintended behaviour. As the system is opaque at runtime, all participants had to spend task time manually revealing sections of the system, with increasing frustration the longer this took to resolve. The opaqueness of the system forced participants to make assumptions about the cause of the unintended behaviour, selecting variables to print to console without any support from the IDE. Making the program state visible in a contextually appropriate way could alleviate these frustrating debugging events, such as a variable window that can show values of contextually relevant variables at runtime.

Build code on a modular foundation

It was observed that 8 out of the 9 participants followed a similar starting-methodology to the task. This methodology was a two-stage process: the first stage was to create a code structure that they can build upon, which P6 described as "scaffolding". The second stage is discussed in the next section: *initiating the creative feedback loop quickly.*

The scaffolding facilitated participants' modular approach to design, offering greater flexibility and extendibility to the work. Some participants, such as P2, rigorously coded their creature in a modular fashion ("If I wanted to implement even more things it would be even harder. So I decided to make it modular."), while P2, P3, and P6 expressed a desire to modularize parts of their code after the task was finished (P3: "If I had more time, I would have separated the creature behaviours into modules").

Initiate the creative feedback loop quickly

As described above, 8 of the 9 participants followed a similar starting-methodology, which comprised of two stages. The second stage involved achieving a minimal representation of the creative goal to the screen: in the case of this task, it was, in almost all cases, an ellipse drawn to the screen which acted as a placeholder for the creature. This initial visual artefact acts as a springboard for ideations, initiating the creative feedback loop.

In all cases observed, this was a considerable milestone reached in the task. We understand this as the creation of a platform for iterated understanding between creative coder and their artistic material. This observed creative feedback loop entered by all participants is described in Figure 2.

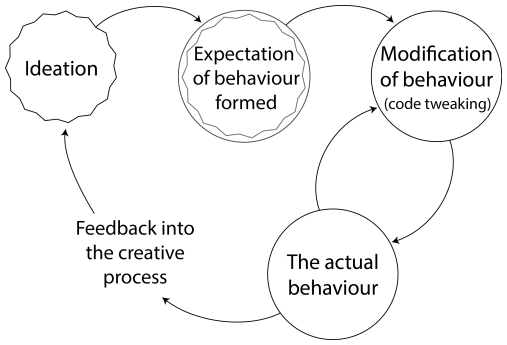

Figure 2. A high level representation of the creative feedback loop present in the process of a creative coder.

Figure 2 describes the process of how creative ideas get manifest as code. The manifestation is the tweak/run/observe cycle.

Allowing this feedback loop to begin uninterrupted at the beginning of a creative task and optimising this tweak/run/observe cycle would increase the role of creative ideation and flow (Csikszentmihalyi, 2009).

Minimise the creative feedback gap

All participants entered multiple phases of a tweak/run/observe cycle. A notable amount of total participant time during the task was spent locked in one of these cycles. The reasons for multiple phases are directly related to the need of *assistance in exploring the space of possible designs*, however the gap that exists between tweaking and observing creates a temporal delay that slows the creative process (McLean and Wiggins, 2010). Beyond slowing the process, the gap can also cripple it. The act of perception is integral to the creative process, and a lag between modification and the perception, judgment and analysis of the outputted result can stifle the creative feedback loop as causal mappings between modification and result are disrupted.

Minimising the gap in this creative feedback loop will alleviate these problems, as well as improve flow.

Assistance in exploring the space of possible designs

All participants engaged in rounds of parametric design tweaking. These were usually rounds of a tweak/run/observe cycle, as explained above. The action is the modification of a parameter of interest, then compiling and running of the program to visually interpret the results. This tweaking of parameters can be considered an exploratory type of creativity, which involves exploring, navigating and testing the potential and boundaries of a conceptual space.

Participants expressed ways this need could be addressed: P1 suggested a grid of possible design avenues to select from. Parameters could be selected, which generate a grid of results, and ideal variations could be selected by the user, evolving the parameter into an optimum state. P4 declared "it would be great to have a live feedback loop", desiring access to parameter variations at runtime.

Maintain a robust mental representation of the software

A surprising discovery in our analysis was the lack of ready-made code utilisation, such as code examples within the Processing environment, relevant code found online, or code the participant is familiar with from a previous project. Utilisation of ready-made code is defined here as a copy and paste command, from the source code into the task project. Contrary to our expectations, ready-made code was utilised by only one participant. This is an unexpected finding, as we hypothesised that ready-made code would be a common starting point for the task across all participants regardless of expertise level. Miryung et al. (2004) state that copy and paste programming is a common strategy of programmers, either from prior work or third-party sources, however only one participant in our study engaged in this action from a source of code which they had no prior familiarity with.

The actions of this participant are interesting as it embodies both the intentions of creative coders and the struggles they face: the participant wished to incorporate a code example titled FractalTree, found on the Processing website. However, after a copy and paste of the code into their working project and a brief peruse, the code was deleted, and the participant started from scratch.

Why the code was deleted is of interest. Even though the code appeared to produce visual properties that the participant wanted, the code was rejected after a brief

perusal of the structure. In the post-task interview, the participant explained why it was rejected, stating that the code was "all math stuff, and I'm no good at that. It's way over my head".

When asked whether the participant would have used the code without the limitation of time, the participant stated that they would, as long as there was a minimal learning curve and it was guaranteed to be relevant to their project.

Surprisingly, only 3 of 9 participants referenced code they were familiar with, either through direct access to previous projects, or searching online for tutorials they have used before and extracting out relevant aspects. Perhaps even more surprising was the lack of any third-party library used by any of the participants.

When questioned in the post-task interview about this lack of third-party library use, 8 out of 9 participants expressed hesitation with utilising third-party libraries. The responses can be categorised into a concern about *maintaining a robust mental representation of the software* they are writing. The time it takes to learn an external library for use in a project requires not only an understanding of the system the library represents, but also the coding style of the library author. These difficulties are only overcome if the participant feels the library is easy to learn, and the library is relevant to their project — a serious concern, as the time and effort placed into understanding a library is considered wasted if it is discovered the library is not relevant, and cannot contribute to the task.

Allowing a creative coder to keep a robust, mental representation of their software at all times affords the integration of external code that lends extensibility to their program, removes the lack of expertise barrier for specialised domains, and affords creative coders to more easily implement their creative ideas. However, achieving this remains an open problem.

DISCUSSION AND FUTURE RESEARCH

The needs presented in this paper have created a set of opportunities for supporting and potentially enhancing creativity in the process of creative coders. These opportunities can be converted into the requirements of a creativity support tool aimed at creative coders. One preliminary opportunity is discussed below.

Towards a creativity support tool for creative coders

All participants reached only a small level of progression during the one hour they had for the task: a base that they could "build more stuff on". 8 of 9 participants expressed a desire to work on the task further if they had more time, incorporating additional visual aesthetics and behaviours.

Specifically addressing one of the needs identified (*build code on a modular foundation*), creativity in this task could be enhanced with modular, hot-swappable creature foundations. This hypothesis is in line with research into implementable types of design patterns, sometimes referred to as templates, or toolkits, that have been suggested to enhance the creativity of programmers. These "building blocks to innovation" (Greenberg, S., 2007) allow programmers to concentrate on creative designs, rapidly generate and test new ideas, innovate concepts, and allow designs to evolve. For example, Lin and Landay (2008) offer a tool for prototyping user interfaces across heterogeneous devices, allowing users to adapt, parameterise and combine based on a collection of design pattern examples that include pre-built user-interface fragments. A user-interface that allows instant selection and use of modular creature foundations would address the need to *build code on a modular foundation*, as well as *maintain a robust mental representation of the software*.

Intended future research involves a more thorough analysis of these needs to extract further creativity support tool requirements. A longitudinal study into the process of creative coders, from beginning a project to completion, utilising ethnographic and contextual inquiry methods, could uncover further unmet needs.

Turning these needs into requirements, a prototype of a creativity support tool can then be evaluated.

REFERENCES

Cinder. (2013). Retrieved 6 Sep 2013, from http://libcinder.org/

Csikszentmihalyi, M. Creativity: Flow and the Psychology of Discovery. HarperCollins (2009).

Greenberg, I., Kumar, D., and Xu, D. Creative coding and visual portfolios for CS1. In Proc. SIGCSE 2012, ACM Press (2012), 247-252.

Greenberg, S. Toolkits and interface creativity. Multimedia Tools 32, 2 (2007), 139-159.

Hewett, T., Czerwinski, M., Terry, M., Nunamaker, J., Candy, L., Kules, B., and Sylvan, E. Creativity support tool evaluation methods and metrics. Creativity Support Tools (2005), 10-24.

Lin, J., and Landay, J.A. (2008). Employing patterns and layers for early-stage design and prototyping of cross-device user interfaces. In Proc. SIGCHI 2008, ACM Press (2008), 1313-1322.

McLean, A., and Wiggins, G.A. Bricolage programming in the creative arts. In Proc. PPIG(2010).

Miryung, K., Bergman, L., Lau, T., and Notkin, D. An ethnographic study of copy and paste programming practices in OOPL. In Proc. ISESE 2004, IEEE (2004), 83-92.

push.conference. (2013). Retrieved 6 Sep 2013, from http://push-conference.com/2013

Reas, C. (2003). {Software} Structures. Retrieved 9 Sep 2013, from http://artport.whitney.org/commissions

Reas, C., and Fry, B. Processing: programming for the media arts. AI & SOCIETY 20,4 (2006), 526-538.

Rosson, M.B., and Carroll, J.M. Active programming strategies in reuse. Springer Berlin Heidelberg (1993).

Shneiderman, B. Creativity support tools. Communications of the ACM 45, 10 (2002), 116-120.

Shneiderman, B., Fischer, G., Czerwinski, M., Myers, B., and Resnick, M. Creativity Support Tools: A workshop sponsored by the NSF (2005).

Turkle, S., & Papert, S. Epistemological Pluralism: Styles and Voices within the Computer Culture. Signs 16, 1 (1990), 128-157.