Imperial College London

COMPUTATIONAL Creativity Group

Computational Creativity Autumn School

Lectures 5 and 6

Philosophical Issues, Guiding Principles and Formalisations

Simon Colton
Computational Creativity Group
Department of Computing
Imperial College, London

ccg.doc.ic.ac.uk

Aims

- To give you a high level framework in which to build your systems for creative purposes
- To get you to think past your code and output, and worry about the impact your project may have
- To give you confidence in undertaking projects where you might one day call your software creative

Overview

- Very high level notions related to Computational Creativity
- An attack on Turing-style tests
- Technical and societal guiding principles for computational creativity
- A first draft of a formalisation for Computational Creativity Theory

Possibly Contentious

1. High Level Notions

Some Difficult Assumptions

- There are no reliable definitions of creativity, in fact such definitions would probably contradict the idea of creativity
- There are no right or wrong processes or methodologies, or good or bad artefacts, in certain domains
- In some domains, the value ascribed by people to generated artefacts is based in part on how they were produced

Science vs. Engineering

- Cognitive sciences approach to computational creativity
  - Study human creativity and try to emulate aspects
  - Use models to further understand human creativity
- Engineering approach to computational creativity
  - Realise that there is no single agreed upon description for multi-faceted human creative behaviour, and that other models of creativity may exists
  - Realise that we can change people's opinions about the notion of creativity and get our software to do this too

Creativity vs. the Perception of Creativity

- People are perfectly capable of performing an ordinary/dull process and because the product of the process has some values not associated with it being creatively produced, those people may pretend that they acted creatively in the process, hence giving us the perception of creativity
  - However, we usually talk about "real creativity" when talking about people, as if it is genuine, and would exist even if no-one was around to notice it happening
- In computational creativity, however, you might find it more useful to talk about the perception people have (or don't have) of creativity in software. I certainly do. In fact, I only care about how people perceive my software, not whether it is actually creative or not (but my language doesn't always reflect this)

A Note about Language

- When people say that a "building is creative", either in scientific or layman situations, they probably mean that they are prepared to project the word *creative* onto the physical and cognitive processes that led to the completion of the building, i.e., that (some of) these processes were different from the norm in some way, or led to a novel concept, and (usually they mean that) the processes produced a building of value. It's unlikely that they mean that the building itself is able to create, but some people talk about *"creativity residing in the artefact"*, which seems like ambiguous nonsense to me.

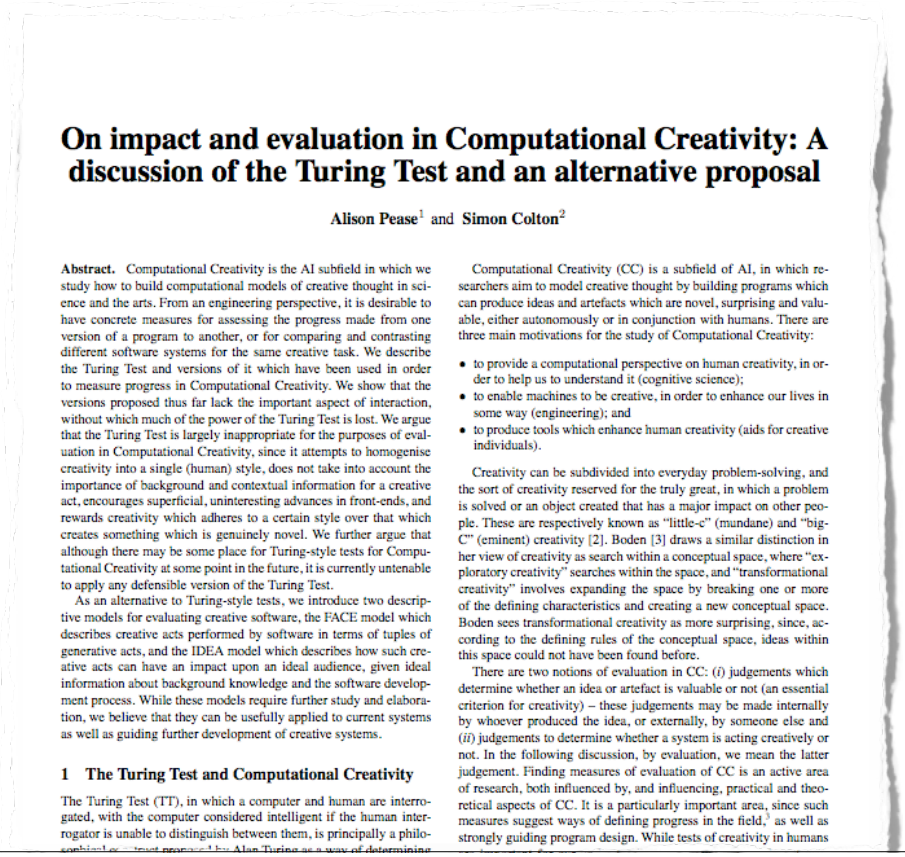
A Note about Language

- Of course, the phrase "the building is creative" is just a harmless shorthand, but it is factually incorrect, and we should strive to reduce ambiguity, especially in scientific writings. We also need to reserve the phrase "creative buildings" for future buildings which are indeed creative, (or for describing buildings in our thought experiments, or in our science fiction books, etc.)
- And yes, we can estimate creativity based on the output of creative acts alone. But, we will likely rely upon default - and probably romantic - assumptions about how people create, and we probably only end up making relative comparisons ("person A has produced better music than person B, hence person A must've been more creative"). These are valid things to do, because person A and B are assumed to have largely similar processes by default, with A innovating somewhat, or with A having a fine-tuned aesthetic sense. However, if we learn of the processes behind the generation of A and B's music, we could perfectly validly change our perception and valuation of their creativity. And this could have an effect of our valuation of their music. This has to be taken seriously into account in computational creativity, where the default notion of the process is uncreative.

Formalisms vs. Implementations

Implementations

Formalisms



On impact and evaluation in Computational Creativity: A discussion of the Turing Test and an alternative proposal

Alison Pease[1] and Simon Colton[2]

Abstract. Computational Creativity is the AI subfield in which we study how to build computational models of creative thought in science and the arts. From an engineering perspective, it is desirable to have concrete measures for assessing the progress made from one version of a program to another, or for comparing and contrasting different software systems for the same creative task. We describe the Turing Test and versions of it which have been used in order to measure progress in Computational Creativity. We show that the versions proposed thus far lack the important aspect of interaction, without which much of the power of the Turing Test is lost. We argue that the Turing Test is largely inappropriate for the purposes of evaluation in Computational Creativity, since it attempts to homogenise creativity into a single (human) style, does not take into account the importance of background and contextual information for a creative act, encourages superficial, uninteresting advances in front-ends, and rewards creativity which adheres to a certain style over that which creates something which is genuinely novel. We further argue that although there may be some place for Turing-style tests for Computational Creativity at some point in the future, it is currently untenable to apply any defensible version of the Turing Test.

As an alternative to Turing-style tests, we introduce two descriptive models for evaluating creative software, the FACE model which describes creative acts performed by software in terms of tuples of generative acts, and the IDEA model which describes how such creative acts can have an impact upon an ideal audience, given ideal information about background knowledge and the software development process. While these models require further study and elaboration, we believe that they can be usefully applied to current systems as well as guiding further development of creative systems.

1 The Turing Test and Computational Creativity

The Turing Test (TT), in which a computer and human are interrogated, with the computer considered intelligent if the human interrogator is unable to distinguish between them, is principally a philo-

Computational Creativity (CC) is a subfield of AI, in which researchers aim to model creative thought by building programs which can produce ideas and artefacts which are novel, surprising and valuable, either autonomously or in conjunction with humans. There are three main motivations for the study of Computational Creativity:

- to provide a computational perspective on human creativity, in order to help us to understand it (cognitive science);
- to enable machines to be creative, in order to enhance our lives in some way (engineering); and
- to produce tools which enhance human creativity (aids for creative individuals).

Creativity can be subdivided into everyday problem-solving, and the sort of creativity reserved for the truly great, in which a problem is solved or an object created that has a major impact on other people. These are respectively known as "little-c" (mundane) and "big-C" (eminent) creativity [2]. Boden [3] draws a similar distinction in her view of creativity as search within a conceptual space, where "exploratory creativity" searches within the space, and "transformational creativity" involves expanding the space by breaking one or more of the defining characteristics and creating a new conceptual space. Boden sees transformational creativity as more surprising, since, according to the defining rules of the conceptual space, ideas within this space could not have been found before.

There are two notions of evaluation in CC: (*i*) judgements which determine whether an idea or artefact is valuable or not (an essential criterion for creativity) – these judgements may be made internally by whoever produced the idea, or externally, by someone else and (*ii*) judgements to determine whether a system is acting creatively or not. In the following discussion, by evaluation, we mean the latter judgement. Finding measures of evaluation of CC is an active area of research, both influenced by, and influencing, practical and theoretical aspects of CC. It is a particularly important area, since such measures suggest ways of defining progress in the field,[3] as well as strongly guiding program design. While tests of creativity in humans

Turing-Style Tests

- Style 1: A dialogue where the point of the exercise is to prove that it would be fair to call your software intelligent
  - Closest to what Turing had in mind
- Style 2: A dialogue where the point of the exercise is to prove that people can't tell the difference to talking to a person and talking to your software
  - So, we implement software which often says unintelligent things
- Style 3: A comparison test with no dialogue, where the point of the exercise is to prove that the output of your software is of a similar (or higher) value to that produced by people
  - This has often been applied in Computational Creativity research

Comparison Tests

- It is certainly a milestone in the development of generative software (and for the field as a whole) if the output can be easily confused with that of people. This is because we can refer to the default position that people act creatively when they produce, and hence it is only fair to describe software similarly, as per my previous point
- And it allows objective comparison, enabling us to show progress in implementations. Importantly, we can be seen to be scientific in our evaluation methodology
- And journalists love setting up Turing-style tests, as it both informs and worries the general public, which helps to sell newspapers...
  - New Scientist and BBC Horizon

However...

- Imagine a comparison test where the tester performs the *reveal:*
- "So, these paintings were painted by recent Royal College of Art graduates"
- "And these ones were painted by.....

  a mass murderer!

- Wouldn't your value judgements change?

Problems 1 and 2

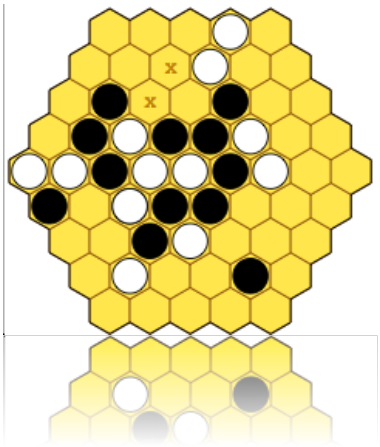
- Turing-style comparison tests set the computer up for a fall
  - The implicit assumption is that software should be very grateful if it is mistaken occasionally for a human
  - So, human level output becomes seen as the only goal of Computational Creativity research
- Software is NOT human!
- So, we end up missing out on possibilities where the software creates valuable, interesting artefacts in non-human ways
  - We should instead be loud and proud about the generative system being computer based, and help people to appreciate the value of computer generated artefacts

Problems 3 and 4

- Turing-style comparison tests massively underestimate the importance of process in certain domains
  - This can lead to alienation of people, certainly in the visual art world, where art theory is all about process
- Turing-style comparison tests answer the wrong question, e.g., which would you prefer, if you had to make up your mind without knowing fully how they were produced
  - Whereas in (commercial/artistic/scientific) reality, we will have full disclosure of practice as well as product
  - Or should we go through this charade with our software for the rest of our lives?

Example: Board Games

- Nestorgames sells a physical version of Yavalath, which was invented by the LUDI system of Cameron Browne
  - There is evidence that the computer invention of the game is a good selling point, which is admittedly unusual
- Nestor is organising a competition, where computer-designed games are blind tested against human-designed games. There will be a large element of publicity in this...
  - But they will sell the games with full disclosure of how they were produced
  - So, are they asking the wrong question?



Problems 5 and 6

- There are no right or wrongs in the visual arts. However, critics can severely inflict pain by saying that your work is "naive" and a "pastiche"
- Turing-style comparison tests might encourage software to act unintelligently, to make it seem more human, hence it could be criticised as naive
- Turing-style comparison tests definitely encourage the generation of pastiche pieces, as the measure of success is whether you have successfully imitated something which isn't you
- Would art graduates be happy if you said their pieces all looked like Monet pictures?

Well put by Alison...

[3]). In [4], Plucker and Makel list "similar, overlapping and possibly synonymous terms for creativity: imagination, ingenuity, innovation, inspiration, inventiveness, muse, novelty, originality, serendipity, talent and unique". The term 'imitation' is simply antipodal to many of these terms.

- Turing-style comparison tests are inappropriate for testing aspects of creative intelligence in software
  - See paper for other arguments

Boden's

"A Turing Test for Artistic Creativity"

In [11], Boden discusses the Turing Test and artistic creativity. She provides an interpretation of the Turing Test which is specifically designed for computer art systems:

> "I will take it that for an 'artistic' program to pass the TT would be for it to produce artwork which was:
>
> 1. indistinguishable from one produced by a human being; and/or
> 2. was seen as having as much aesthetic value as one produced by a human being." [11, p. 409]

[11] M. A. Boden. The Turing test and artistic creativity. *Kybernetes*, 39(3):409–413, 2010.

Boden's

"A Turing Test for Artistic Creativity"

Boden describes several systems which produce art or music, which she considers to be either non-interactive or unpredictably interactive (such as a piece of art which responds to audience members or participants in ways they do not understand). She discusses comparisons with both mediocre human art, in this case pastiches of given styles (perhaps comparable to work by an art student exploring a given style), as well as examples which match world class human art, of interest as an artwork in itself (comparable to work done by a practising artist). She argues that the following systems all pass (her version of) the TT:

- Richard Brown's Starfish
- Harold Cohen's AARON
- Art by Boden and Edmunds
- David Cope's EMI

Boden's

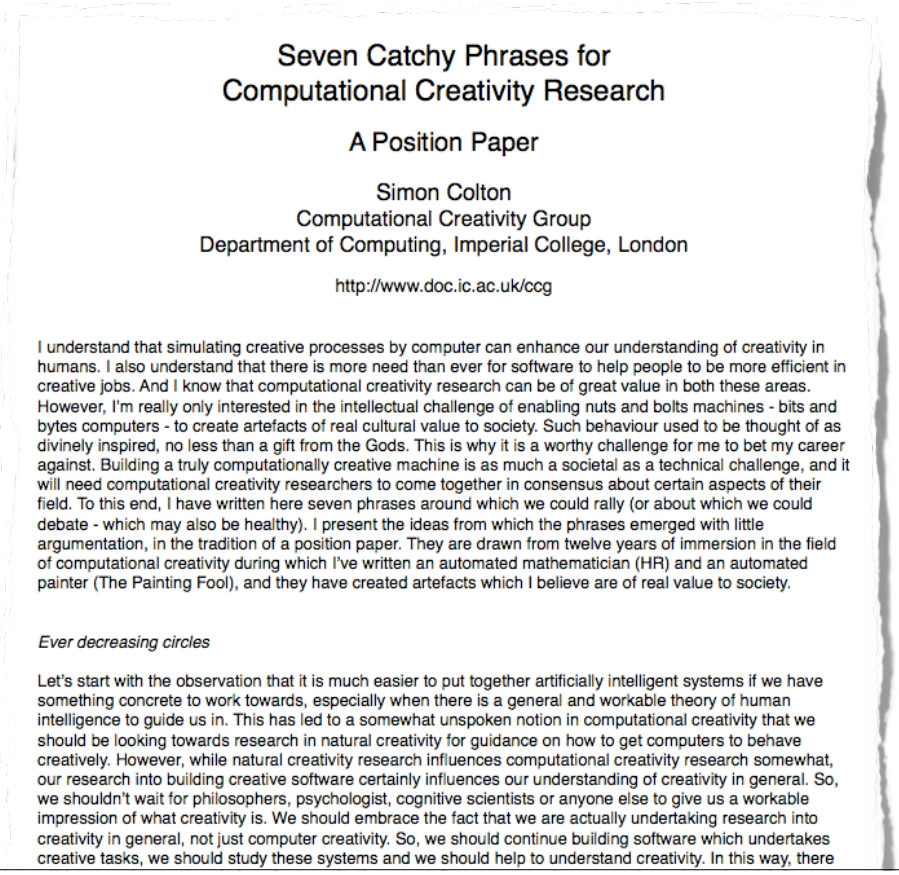
"Turing Test for Artistic Creativity"

In particular, Boden argues that "If being exhibited alongside Rothko, in a 'diamond jubilee' celebration of these famous artists, does not count as passing the Turing Test, then I do not know what would." [11, p. 410].

Our Objections...

- It's an interpretation of Turing's test which bears little resemblance to the original idea
  - There is no dialogue or interaction of any kind with the system as part of the test
  - The test can be passed without comparison to human intelligence, or even human output
- So, it's possible to pass the originally conceived Turing test (testably achieving human-level intelligence), yet not pass Boden's test
  - Yet - as evidenced by the Starfish and by Boden and Edmunds art - it's possible to pass Boden's test without exhibiting any higher level cognitive functions

The Starfish...



Seven Catchy Phrases for
Computational Creativity Research

A Position Paper

Simon Colton
Computational Creativity Group
Department of Computing, Imperial College, London

http://www.doc.ic.ac.uk/ccg

I understand that simulating creative processes by computer can enhance our understanding of creativity in humans. I also understand that there is more need than ever for software to help people to be more efficient in creative jobs. And I know that computational creativity research can be of great value in both these areas. However, I'm really only interested in the intellectual challenge of enabling nuts and bolts machines - bits and bytes computers - to create artefacts of real cultural value to society. Such behaviour used to be thought of as divinely inspired, no less than a gift from the Gods. This is why it is a worthy challenge for me to bet my career against. Building a truly computationally creative machine is as much a societal as a technical challenge, and it will need computational creativity researchers to come together in consensus about certain aspects of their field. To this end, I have written here seven phrases around which we could rally (or about which we could debate - which may also be healthy). I present the ideas from which the phrases emerged with little argumentation, in the tradition of a position paper. They are drawn from twelve years of immersion in the field of computational creativity during which I've written an automated mathematician (HR) and an automated painter (The Painting Fool), and they have created artefacts which I believe are of real value to society.

Ever decreasing circles

Let's start with the observation that it is much easier to put together artificially intelligent systems if we have something concrete to work towards, especially when there is a general and workable theory of human intelligence to guide us in. This has led to a somewhat unspoken notion in computational creativity that we should be looking towards research in natural creativity for guidance on how to get computers to behave creatively. However, while natural creativity research influences computational creativity research somewhat, our research into building creative software certainly influences our understanding of creativity in general. So, we shouldn't wait for philosophers, psychologist, cognitive scientists or anyone else to give us a workable impression of what creativity is. We should embrace the fact that we are actually undertaking research into creativity in general, not just computer creativity. So, we should continue building software which undertakes creative tasks, we should study these systems and we should help to understand creativity. In this way, there

Guidelines

- The idea is to possibly appeal to these guidelines during the engineering, testing and engagement parts of your project
- But also, they're here to get you thinking about some more of the philosophical aspects of Computational Creativity research

1. Ever decreasing circles

- It's important to recognise that we have the potential to contribute as much to the understanding of human creativity as psychological studies do
- We don't necessarily have to wait for discoveries about the nature of human creativity to add creative behaviour to our software
- We can imagine mutual benefits where all fields learn from each other - spiralling down to the truth

2. Paradigms lost

- As AI researchers and practitioners, you don't necessarily have to see every intelligent task as a problem solving exercise
- If you do apply a reductionist approach, remember to put the pieces back together again
- The artefact generation paradigm can be found again: intelligent tasks are framed as opportunities to generate something of cultural value

3. The whole is more than a sum of the parts

- It is often more difficult to get your software to talk to other software than to implement a pale version of the software you want
- However, it's likely that your software will be more powerful if you join forces with others
- And it helps to attract people to Computational Creativity if we use their software

4. Climbing the meta-mountain

- We need to constantly ask ourselves how we can hand over creative responsibilities to the software
- Plan in advance to one day get the software to take over what you are doing in projects
- In particular, think about how the software can take on aethetic responsibilities, and possibly show intentionality in its work
- Try and hand over meta-level control and climb the mountain to the top

5. The creativity tripod

- People often take details of a generative process into account when they valuate output artefacts
- The default position in public perception is that software cannot be creative, which can lead to a vicious circle where output is never seen as valuable
  - Hence, we need to manage this public perception
- People will generally not ascribe creativity to software if it is lacking skill, appreciation or imagination. So, we can be proactive and aim to implement behaviours which tick these boxes
- Remember that tripods have three legs, with three sections to each leg: (programmer,/user software, audience)

6. Beauty is in the mind of the beholder

- Value is not just skin-deep
- If you aim for pastiche, you might get useful software, but it's unlikely to ever be taken seriously as creative in its own right
- Think about the process/output of/from your software having an impact on people, rather than the imitation game
- Ask yourself: "Is a Turing-style test the right way to assess your software?" - people need to know about the entire creative act if they are to assess the output

7. Good art makes you think

- The output of creative software should really be seen as an invitation to start a dialogue
- Decorative art has value, but it is unlikely to be seen as great art, because it doesn't give people an opportunity to have a dialogue with the artwork
  - Dialogues can be audience-centric, or involve cultural aspects of the day, historical concepts, etc.
- Our flavour of AI makes people think more rather than less

Computational Creativity Theory:
The FACE and IDEA Descriptive Models

Simon Colton[1], **John Charnley**[1] **and Alison Pease**[2]

[1] Computational Creativity Group, Department of Computing, Imperial College, London, UK. ccg.doc.ic.ac.uk

[2] School of Informatics, University of Edinburgh, UK.

Abstract

We introduce computational creativity theory (CCT) as an analogue in computational creativity research to computational learning theory in machine learning. In its current draft, CCT comprises the FACE descriptive model of creative acts as tuples of generative acts, and the IDEA descriptive model of the impact such creative acts may have. To introduce these, we simplify various assumptions about software development, background material given to software, how creative acts are performed by computer, and how audiences consume the results. We use the two descriptive models to perform two comparisons studies, firstly for mathematical discovery software, and secondly for visual art generating programs. We conclude by discussing possible additions, improvements and refinements to CCT.

Introduction

Machine learning is a very well established research area, and has possibly had the greatest impact of all AI subfields, with successful applications to classification and prediction tasks in hundreds of areas of discourse. Machine learning researchers asked: "What does it mean to say that a computer has *learned* something?" In computational creativity research, we have similarly asked: "What does it mean to say that a computer has *created* something?" It is therefore sensible to look at how machine learning developed to suggest future directions for computational creativity research. Human learning involves many behaviours, including memorisation, comprehension, abstraction and generalisation. While textbooks like (Mitchell 1997) use general terms such as "learning how to do something by observing it being done", in reality, machine learning researchers have largely studied how to induce a concept definition (or set of

putational learning theory (CLT). In (Angl
Angluin states that the goals of CLT are to:

Give a rigorous, computationally de
plausible account of how learning ca

Each word in this statement has been care
particular, by requiring *computational deta*
lights that CLT discusses actual algorithms
requiring *rigour*, he stresses that the purp
state and prove theorems about the nature
algorithms. Such theorems deal with learn
PAC model (Valiant 1984), i.e., which type
be induced by which methods, and with ef
extent to which certain methods must sea
mentioning the *plausibility* of the account
grounds CLT to describing machine learni
with reasonable computational resources, b
mation and interaction requirements. Fina
the account to one which describes how
done, Angluin emphasises that CLT does n
encompass approaches to computer learnir
fers a particular formalism which researche
sess progress. The benefits of CLT have be
via theoretical results being turned into prac
most notably the introduction of ensemble
such as boosting (Schapire 1990), whereby
tors are assembled from multiple weak one

We introduce here the first draft of a
computational creativity theory (CCT). In t
aim for CCT to provide a rigorous, computa
and plausible account of how creation can
step has been to provide two plausible de
which can be used to describe the processir
of software designed to exhibit

Computational Creativity Theory:
Inspirations behind the FACE and IDEA descriptive models

Alison Pease[1] **and Simon Colton**[2]

[1] School of Informatics, University of Edinburgh, UK.

[2]Computational Creativity Group, Department of Computing, Imperial College, London, UK. ccg.doc.ic.ac.uk

Abstract

We introduce two descriptive models for evaluating creative software; the FACE model, which describes creative acts performed by software in terms of tuples of generative acts, and the IDEA model, which describes how such creative acts can have an impact upon an audience, given information about background knowledge and the software development process. We show how these models have been inspired both by ideas in the psychology of creativity and by an analysis of acts of human creativity.

Introduction

To enable the Computational Creativity (CC) community to make objective, falsifiable claims about progress made from one version of a program to another, or for comparing and contrasting different software systems for the same creative task, we need concrete measures of evaluation. There are two notions of evaluation in CC: (*i*) judgements which determine

F^p: a method for generating framing information
F^g: an item of framing information
A^p: a method for generating aesthetic measures
A^g: an aesthetic measure
C^p: a method for generating concepts
C^g: a concept
E^p: a method for generating expressions of a concept
E^g: an expression of a concept

Any particular creative episode can be expressed in terms of at least one of these components (it may well be the case that not all of the components will be present). Note that items of framing information, F^g, may refer to any combination of A^p, A^g, C^p, C^g, E^p, and E^g. In order to cover as many creative acts as possible, we assume only that there must be something new created for the question of creativity to arise. This could be very small, a brush stroke of an artist, an inference step by a mathematician, a single note in a piece of music. Our model, then, covers "merely generative" acts as well as "fundamentally generative" acts. By drawing our baseline at "merely generative", our model can be used to describe the most basic "creative act" possible, we avoid the thorny issue

Aspirations for

Computational Creativity Theory

- Aim for computational learning theory:
  - "To give a rigorous, computationally detailed and plausible account of how learning can be done"

(Dana Angluin)

Aspirations for

Computational Creativity Theory

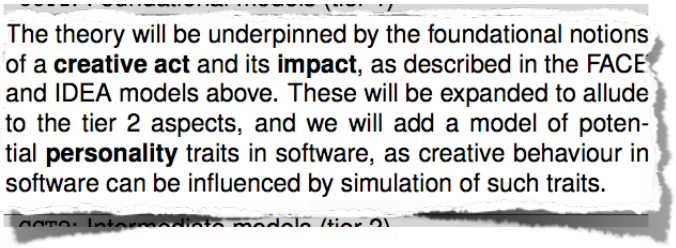
Aim is to prove theorems about the nature of software, to enable comparisons

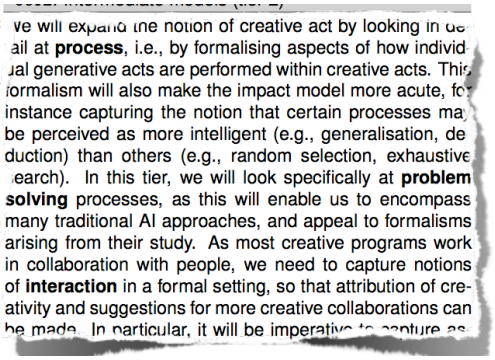
- Aim for computational creativity theory:
  - "To give a rigorous, computationally detailed and plausible account of how creativity can be done"

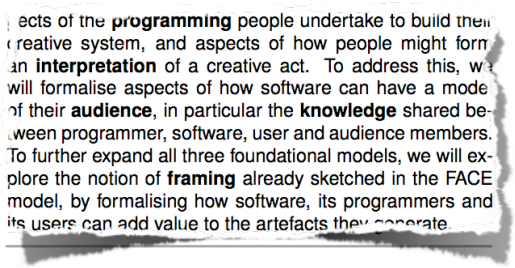
Ground the theory in reality with respect to the amount of resources, user interaction, etc.

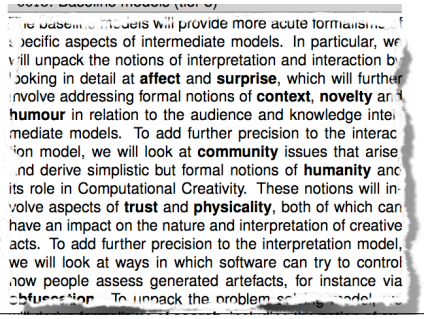
Not aiming to capture all senses in which software can create, but be a rallying point

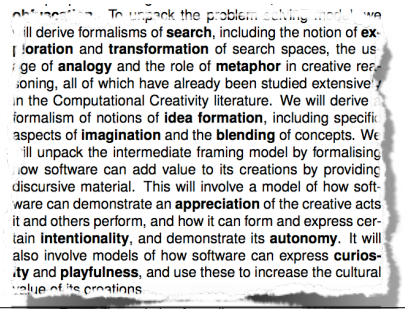
Tiers of words


The theory will be underpinned by the foundational notions of a **creative act** and its **impact**, as described in the FACE and IDEA models above. These will be expanded to allude to the tier 2 aspects, and we will add a model of potential **personality** traits in software, as creative behaviour in software can be influenced by simulation of such traits.


ail at **process**, i.e., by formalising aspects of how individ
ual generative acts are performed within creative acts. This
formalism will also make the impact model more acute, fo
instance capturing the notion that certain processes ma
be perceived as more intelligent (e.g., generalisation, de
duction) than others (e.g., random selection, exhaustive
search). In this tier, we will look specifically at **problem solving** processes, as this will enable us to encompass many traditional AI approaches, and appeal to formalisms arising from their study. As most creative programs work in collaboration with people, we need to capture notions of **interaction** in a formal setting, so that attribution of creativity and suggestions for more creative collaborations can be made. In particular, it will be imperative to capture as


ects of the **programming** people undertake to build their
reative system, and aspects of how people might form
an **interpretation** of a creative act. To address this, we
will formalise aspects of how software can have a mode
of their **audience**, in particular the **knowledge** shared be-
tween programmer, software, user and audience members.
To further expand all three foundational models, we will ex-
plore the notion of **framing** already sketched in the FACE
model, by formalising how software, its programmers and
its users can add value to the artefacts they generate.


ecific aspects of intermediate models. In particular, we
will unpack the notions of interpretation and interaction by
looking in detail at **affect** and **surprise**, which will further
involve addressing formal notions of **context**, **novelty** and
humour in relation to the audience and knowledge inter-
mediate models. To add further precision to the interac
ion model, we will look at **community** issues that arise
nd derive simplistic but formal notions of **humanity** and
its role in Computational Creativity. These notions will in-
volve aspects of **trust** and **physicality**, both of which can
have an impact on the nature and interpretation of creative
acts. To add further precision to the interpretation model,
we will look at ways in which software can try to control
how people assess generated artefacts, for instance via


ill derive formalisms of **search**, including the notion of **ex-**
loration and **transformation** of search spaces, the us
ge of **analogy** and the role of **metaphor** in creative rea-
soning, all of which have already been studied extensively
in the Computational Creativity literature. We will derive
formalism of notions of **idea formation**, including specific
aspects of **imagination** and the **blending** of concepts. We
will unpack the intermediate framing model by formalising
how software can add value to its creations by providing
discursive material. This will involve a model of how soft-
ware can demonstrate an **appreciation** of the creative acts
it and others perform, and how it can form and express cer-
tain **intentionality**, and demonstrate its **autonomy**. It will
also involve models of how software can express **curios-**
ity and **playfulness**, and use these to increase the cultural
value of its creations.

Descriptive Models

Should Provide...

- Some simplifying assumptions related to programming/ running software and the appreciation by an audience of its behaviour and its output
- A set of conceptual definitions which can be used to describe behaviour in software/programmers/audiences associated with acts of creation
- A set of concrete calculations based on the definitions, which can be used to compare and contrast different software systems
- Some suggesions for how the calculations could be applied in different application domains

The FACE model

To describe creative acts performed by software

- Simplifying assumptions:
  - Even the smallest generative act can be described as a creative act (e.g., multiplying two numbers together)
    - Independently of the amount of impact the act might have
  - We can effectively restrict ourselves to discussing how software can produce eight types of output
  - Both the processes performed by software and the results of the processing need to be covered
  - The quality and quantity of creative acts can be used to compare creative software

The FACE model

To describe creative acts performed by software

process level, where new ways to generate and assess arte-facts are invented. We define a *creative act* as a non-empty tuple of generative acts. Each tuple contains exactly zero or one instance of eight types of individual generative act. The different types of generative act are denoted by the following letters with g or p superscripts, representing generative acts which produce:

E^g:	an expression of a concept
E^p:	a method for generating expressions of a concept
C^g:	a concept
C^p:	a method for generating concepts
A^g:	an aesthetic measure
A^p:	a method for generating aesthetic measures
F^g:	an item of framing information
F^p:	a method for generating framing information

- We use lower case to denote the output from the individual generative acts in the creative act tuples, and a bar notation to indicate constituent generative acts performed by a third party

The FACE model

To describe creative acts performed by software

- Comparison methods:
  - Volume of creative acts
  - Ordering of creative acts, e.g., <A^g, C^g, E^g> deemed more creative than <C^g, E^g>
  - By the nature of the processes, e.g., random deemed less creative than inductive
- By using the aesthetic function (given or invented) in a domain

ing such a session of creative acts:

$$average(S) = \frac{1}{n}\sum_{i=1}^{n} \overline{a^g}(c_i^g, e_i^g)$$
$$best_ever(S) = max_{i=1}^{n}\left(\overline{a^g}(c_i^g, e_i^g)\right)$$
$$worst_ever(S) = min_{i=1}^{n}\left(\overline{a^g}(c_i^g, e_i^g)\right)$$
$$precision(S) = \frac{1}{n}|\{(c_i^g, e_i^g) : 1 \leq i \leq n \wedge \overline{a^g}(c_i^g, e_i^g) > t\}|$$
$$reliability(S) = best_ever(S) - worst_ever(S)$$

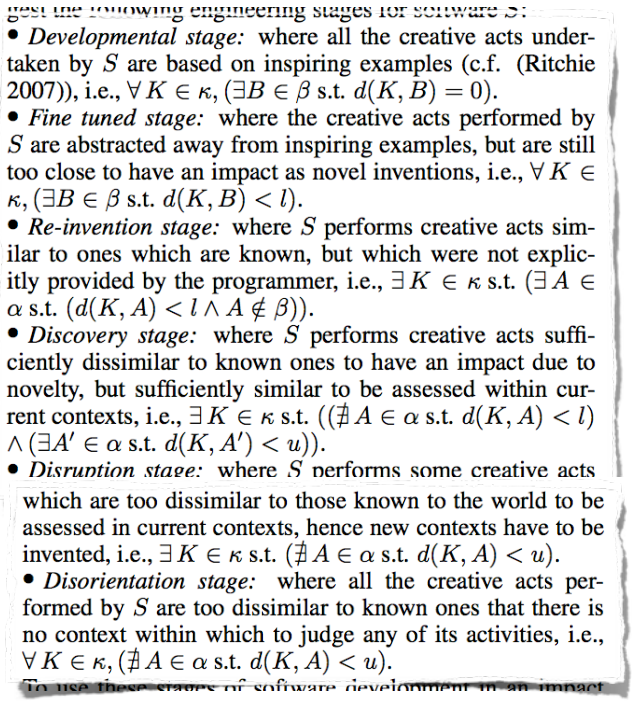
The IDEA model

To describe the impact that creative acts may have

- Motivations
  - Creative software can invent its own aesthetics, so we need to generalise past value judgements
  - The influence of the programmer/user has to be assessed to evaluate the impact caused by the behaviour of the software
- Simplifying assumptions
  - An ideal software development process described by FACE-tuples
  - Full knowledge of the creative acts that went into the production of all the relevant background knowledge
  - An ideal audience of members, m, able to perfectly assess their appreciation of creative acts, A, along two axes:
    - Well being: $wb_m(A)$ and cognitive effort: $ce_m(A)$ [Note not creativity directly]

The IDEA model

To describe the impact that creative acts may have



- *Developmental stage:* where all the creative acts undertaken by S are based on inspiring examples (c.f. (Ritchie 2007)), i.e., $\forall K \in \kappa, (\exists B \in \beta \text{ s.t. } d(K,B) = 0)$.
- *Fine tuned stage:* where the creative acts performed by S are abstracted away from inspiring examples, but are still too close to have an impact as novel inventions, i.e., $\forall K \in \kappa, (\exists B \in \beta \text{ s.t. } d(K,B) < l)$.
- *Re-invention stage:* where S performs creative acts similar to ones which are known, but which were not explicitly provided by the programmer, i.e., $\exists K \in \kappa \text{ s.t. } (\exists A \in \alpha \text{ s.t. } (d(K,A) < l \wedge A \notin \beta))$.
- *Discovery stage:* where S performs creative acts sufficiently dissimilar to known ones to have an impact due to novelty, but sufficiently similar to be assessed within current contexts, i.e., $\exists K \in \kappa \text{ s.t. } ((\nexists A \in \alpha \text{ s.t. } d(K,A) < l) \wedge (\exists A' \in \alpha \text{ s.t. } d(K,A') < u))$.
- *Disruption stage:* where S performs some creative acts which are too dissimilar to those known to the world to be assessed in current contexts, hence new contexts have to be invented, i.e., $\exists K \in \kappa \text{ s.t. } (\nexists A \in \alpha \text{ s.t. } d(K,A) < u)$.
- *Disorientation stage:* where all the creative acts performed by S are too dissimilar to known ones that there is no context within which to judge any of its activities, i.e., $\forall K \in \kappa, (\nexists A \in \alpha \text{ s.t. } d(K,A) < u)$.

- Need a distance function, d, to tell how close two creative acts are
- Formalism for the development of creative software with respect to the programmer/user's influence
- Compare software in terms of its autonomy from the programmer and from the cultural context it was programmed within

The IDEA model

To describe the impact that creative acts may have

$$dis(A) = disgust(A) = \frac{1}{2n}\sum_{i=1}^{n}(1 - wb_i(A))$$
$$div(A) = divisiveness(A) = \frac{1}{n}\sum_{i=1}^{n}|wb_i(A) - m(A)|$$
$$ind(A) = indifference(A) = 1 - \frac{1}{n}\sum_{i=1}^{n}|wb_i(A)|$$
$$pop(A) = popularity(A) = \frac{1}{2n}\sum_{i=1}^{n}(1 + wb_i(A))$$
$$prov(A) = provocation(A) = \frac{1}{n}\sum_{i=1}^{n}(ce_i(A))$$

$$acquired_taste(A) = (pop(A) + prov(A))/2$$
$$instant_appeal(A) = (1 + pop(A) - prov(A))/2$$
$$opinion_splitting(A) = (1 + div(A) - prov(A))/2$$
$$opinion_forming(A) = (div(A) + prov(A))/2$$
$$shock(A) = (1 + dis(A) - prov(A))/2$$
$$subversion(A) = (dis(A) + prov(A))/2$$
$$triviality(A) = (1 + ind(a) - prov(A))/2$$

- Formalism attempting to capture some common notions of impact, using the well-being and cognitive effort measures of the ideal audience
- m(A) is the mean well being amongst the ideal audience

Comparison Study

Mathematical Discovery Software

- Comparison of types of creative act
  - AM and HR: $\langle \overline{A}^g, C^g, E^g \rangle$
    - But HR has more types of C^g and E^g generative acts
  - Meta-HR: $\langle C^p, C^g, E^g \rangle$ and $\langle A^g, C^g, E^g \rangle$
  - TM took ModGen from $\langle E^g \rangle$ to $\langle C^g, E^g \rangle$
- In terms of precision, AM outperforms HR, but AM never left the fine-tuned stage of development, whereas we argue that HR is in the discovery stage, hence has had more impact

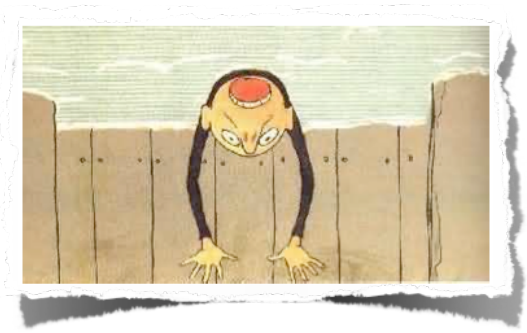
Comparison Study

Art Generation Software

- Comparison of types of creative act
  - AARON and The Painting Fool: $\langle C^g, E^g \rangle$
    - But The Painting Fool has more types of C^g
  - The Painting Fool collage generation: $\langle A^g, C^g, E^g \rangle$
  - TPF + HR fitness function invention:
    - $\langle A^g, C^g, E^g \rangle$ = <fitness function, scene, rendering>
- Most evolutionary art systems: $\langle \overline{A}^g, C^g, E^g \rangle$, but NEvAr performs creative acts of the form: $\langle F^g, \overline{A}^g, C^g, E^g \rangle$ because it uses mathematical fitness functions

See Alison's Paper for...

- Motivations for the FACE and IDEA models coming from cognitive science, psychology and philosophy
- Some links to existing Computational Creativity formalisms, such as from Ritchie, Wiggins, etc.
- Case studies from the history of mathematics and the visual arts



Next Steps...

- Produce more comparison studies using the formalisms, especially in musical and linguistic domains (with your help...?)
  - Fix the problems in CCT which arise
- Relate the theory more to existing formalisms
  - From Computational Creativity, but also by expanding CLT
- Add descriptive models to CCT
  - I'm particularly interested in the ways in which software can obfuscate what it has done and what it has produced
    - Theory might drive practice in this instance

Next Steps...

In order to study creativity, and in particular to implement creative software, it is important to de-mystify creative processes. However, we argue that, in many senses, adding mystery (or allowing it to persist) is an inherent part of the creative process.

This has led to the development of the XXX descriptive model which forms part of Computational Creativity Theory. This model can be used to compare and contrast software in terms of the value they gain by productive use of mystery in their behaviour. We motivate the model by argumentation and appeals to the literature, then introduce various formalisations and show how these could be used to show progress in certain domains of application.

Next Steps...

- "The Programmer's Programmer"
  - Your software development environment will act as a creative partner in building software
- Based on theoretical studies where CCT measures are applied to small changes in code
- Practically: multi-core machine which is constantly making possible changes to your software, compiling it, and showing you the results
  - You can quickly choose a new path to go down

The Take Home...

- The discussions you've seen between the lecturers here (in public and in the pub) highlight that this is a great time to get involved in helping us to define aspects of our field through formalisations of notions of creativity in software
  - We've come a long, long way. But there is still a long, long way to go

Questions?