The creative turn: new challenges for computing

Andrew Hugill* and Hongji Yang

Creative Computing Cluster,
Bath Spa University,
Newton Park, Bath, BA2 9BN, England, UK
E-mail: a.hugill@bathspa.ac.uk
E-mail: h.yang@bathspa.ac.uk
*Corresponding author

Abstract: This is a call to action. The time is ripe for a creative turn in computing. This article sets out a vision of what Creative Computing is and what it might become. It distinguishes between Creative Computing and computational creativity, and then lays out a theoretical framework. It proposes that Creative Computing is mainly happening in software. It compares the process of software development with the process of artistic creation and looks at ways in which the two might productively overlap or merge. It considers the levels of abstraction required in both and interrogates both their semantics and underlying principles from a philosophical and practical point of view. Finally, it sets three types of new challenges for Creative Computing in terms of software.

Keywords: creative; computing; challenges; abstraction; levels; composition; engineering; creativity; turn.

Reference to this paper should be made as follows: Hugill, A. and Yang, H. (2013) 'The creative turn: new challenges for computing', *Int. J. Creative Computing*, Vol. 1, No. 1, pp.4–19.

Biographical notes: Andrew Hugill is the Director of the Creative Computing Cluster at Bath Spa University. He has been working in digital creativity since the 1980s. He is a Research Professor and composer. He is a panel member of the European Research Council. His book *The Digital Musician* (Routledge 2008, 2012) is a standard text on most music technology degree programmes. Other monographs include *Pataphysics: A Useless Guide* (MIT Press, 2012). His current research includes developing a creative search engine, and writing an online opera in collaboration with the The Opera Group. He is a National Teacher Fellow of the Higher Education Awards and was highly commended in the 2006 Times Higher Education Awards for 'the most imaginative use of distance learning'.

Hongji Yang is the Deputy Director of the Creative Computing Cluster at Bath Spa University. He obtained his PhD degree at Durham University. His research interests cover computer organisation, networking, software engineering and recently creative computing, and he has published in all these areas (five books and well over 300 refereed papers). He was the Deputy Technical Director for the Software Technology Research Laboratory at De Montfort University, which accommodated over 100 PhD students. He has been an organiser for several leading international conferences, such as the IEEE International Conference on Software Maintenance and the IEEE Computer Software and Application Conference. It is planned that he will chair the IEEE International Symposium of Creative Computing 2014 in Oxford. He is devoted to developing current research into conducting computing in a creative way.

1 Creative Computing

It is important to differentiate between the terms 'Creative Computing' and 'computational creativity'. Intuitively the former is about doing computations in a creative way, while the latter is about achieving creativity through computation. You can think of the latter falling into the artificial intelligence category (using formal computational methods to mimic creativity as a human trait) (Colburn and Shute, 2007) and the former being a more poetic endeavour of how the computing itself is done, no matter what the actual purpose of the programme might be (Hugill et al., 2013).

Creative Computing seeks to reconcile the objective precision of computer systems (mathesis) with the subjective ambiguity of human creativity (aesthesis). This underlying tension has a profound impact upon the creation of digital culture itself, in ways that are frequently unacknowledged or poorly understood. In many cases, artists and people working in the humanities accept the unambiguous constraints of computer systems because they have the appearance of a neutral authority, of scientific 'fact'. However, computers are created by people, and there is no reason why they cannot better adapt to serve the needs of the creative community.

Although human beings increasingly turn to computers as aids to creativity, the way the software is engineered frequently enforces compromise or, worse, inhibits creativity through unwelcome constraints. This is not to deny the potential creative benefits of constraints, but is simply a recognition of the relative lack of sophistication of existing software. Nor is this necessarily an argument for improved simulation, a form of machine intelligence that takes us closer to the model of the human brain (although simulation may prove to be one of the key aspects of the evolution of Creative Computing). It is rather a challenge for software to become a more effective servant of people by being more adaptive, smarter and better engineered to cope with frequent changes of direction, inconsistencies, irrelevancies, messiness and all the other vagaries that characterise the creative process.

In his Strachey Lecture to the University of Oxford, Anthony Finkelstein of University College London, identified ten 'Open Challenges' in software engineering which come close to the kinds of issues that drive Creative Computing. These include a move away from current anecdotal practices towards an 'evidence-based' approach that engages with social media more effectively, a weaving together of architectures and requirements (what Finkelstein calls 'twin peaks') to produce detailed integrated specifications, and a convergence of web and software engineering standards that can handle large scale and rapid variations in demands (Finkelstein, 2011). Clarke et al. also identify a need to improve engineering practices by developing software that evaluates, changes and improves itself and by "making N people N times as productive as one" (Clarke et al., 1999). We would substitute the word 'creative' for 'productive'. Finally, Broy (2006) makes the point that "tackling the daunting challenges of complex software systems development requires a broad stream of research supported by several new technical competencies, including a good understanding of system modeling, the effective use of models, and a modelling theory of discrete event systems".

Computing, in its search for new challenges, is increasingly addressing the question of how best to serve human creativity. Whereas computing traditionally crosses various levels of abstraction in its path from user requirements (high level) to machine language (low level), the challenge facing Creative Computing is to do so creatively. This represents a new level of sophistication for computing. Current creative applications are ones to which creative people must adapt if they are to realise the potential they contain. All too often, this involves compromise and simplification of the creative idea. The kind of disambiguation required for effective computing affects precisely that ambiguity which makes creativity worthwhile. This has an obvious semantic consequence, but is more profoundly located at the level of system design and engineering.

The need for this creative turn is clear. Modern problems are too complex for human beings without tools to solve. The solutions are transdisciplinary; they invariably require input from several disciplines. Computing is the common factor that links these disciplines. Laboratory studies have only limited relevance, because computer simulations are generally preferable. The natural sciences are also insufficient as they reach the limits of the knowable and necessarily move into purely digital domains. Nor are such limitations confined only to the sciences. The arts and humanities, too, address large and complex questions that are unknowable by a single scholar. Collaboration is becoming the modern mode for studying these complex questions. In all cases, it is computing that substrates human scholarship and enquiry. Yet, the computer systems themselves have inherent limitations that are the consequence of their inability to respond creatively to the challenges presented to them. This is the objective of Creative Computing: to engineer systems that may provide satisfactory creative tools for users in all spheres of human endeavour.

To achieve this requires an increased understanding of human creativity and collaboration between creative people and software engineers. In particular, a mapping of the creative process needs to be undertaken in such a way that does not compromise the layers of ambiguity and uncertainty that are its main characteristics. This mapping will provide the basis for the reformulation of an approach to software engineering that will reflect this in vivo knowledge. In particular, software applications that continuously rewrite themselves in real time in response to the creative needs of the particular problem or question will be the primary outputs of Creative Computing. Advances towards this ideal are already being made, but there is still a great distance to travel. Previous theory has some value in helping to identify the best ways forward, but also some key weaknesses that the creative turn will address.

2 The creative turn

The creative turn will be both *local* and *global.* To take one illustrative example of this phenomenon: on 12th November 2011, Mr. Liu Binjie, Director of Press and Publication Administration for the Chinese Communist Party observed during the China Copyright Annual Meeting in Beijing that the creativity component of many domestic cultural and artistic products is very low, with more than 90% being copies. This perception of a creative lack reflected the Chinese Government's identification of creativity as a national priority. As Michael Keane has argued, their aspiration is to replace the familiar label 'Made in China' with 'Created in China':

> "A great new leap forward is imminent. The 'world factory' is no longer the default setting for development. China aspires to be a serious contender for the spoils of the global cultural and service economies." (Keane, 2007)

Western cultures have demonstrated the value of creativity over many centuries, and are consequently seen to have a 'head start' on the Chinese. Nevertheless, it is realistic to predict that their efforts will rapidly bear fruit, both at the local and global levels. These fruits will, by definition, be both novel and surprising. Given the high level of Chinese expertise in engineering, it also seems likely that new approaches to software engineering will be part of the process towards more creative practice.

These local and global levels reflect the two types of creativity identified by Boden (2003):

- *P-creativity* (short for psychological creativity), which is the personal kind of creativity that is novel in respect to the individual mind
- *H-creativity* (short for historical creativity), which is fundamentally novel in respect to the whole of human history.

Boden says that creative ideas are *surprising* because they go against our expectations. "The more expectations are disappointed, the more difficult it is to see the link between old and new" (Boden, 2003). This suggests that fewer expectations (an open mind) allow creativity to happen more easily. Empirical experiences (Ritchie, 2007) form expectations, which hinder our ability to accept creative ideas when they happen. In order to be able to recognise creative ideas we need to be able to see what they all have in common and in what way they differ and not reject unusual, unexpected ones:

> "Unless someone realises the structure which old and new spaces have in common, the new idea cannot be seen as the solution to the old problem. Without some appreciation of shared constraints, it cannot even be seen as the solution to a new problem intelligibly connected with the previous one." (Boden, 2003)

This is in essence the challenge that faces Creative Computing, whether in China or elsewhere. The shared constraints that need to be appreciated concern crossing levels of abstraction, tools and environments, user requirements and acceptance, and so on. On close examination, these are a matter of considerable disagreement within computing, both from a formal or theoretical perspective and from a more pragmatic, engineering point of view. Yet, they are precisely the old problems which require a new solution. The lack of realisation of this structure has consequently enforced ignorance of the creative aspects of software engineering.

There is today a proliferation of more or less effective 'apps' that are designed to support and enable creative activities. But there is much more that could be done both to engineer software in a more creative way and, most importantly, to place sophisticated creative tools at the service of humans. In the present situation, people often (even always) have to adapt their practices to suit the software, rather than the other way around, e.g., people need to code in a computing language that a machine can understand before a task can be computed. The challenges posed by creativity have been generally overlooked, yet they are among the most important facing humanity. Creative solutions to complex problems are widely agreed to be the most desirable, but where are the tools that are specifically designed to help achieve these?

3 Creative Computing: a software challenge

We argue that Creative Computing is almost exclusively a software engineering challenge, because of its position within the traditional 'Three Paradigms' of computing (Eden, 2007):

- The *rationalist* paradigm, which was common among theoretical computer scientists, defines computer science as a branch of mathematics, treats programmes on a par with mathematical objects, and seeks certain, a priori knowledge about their 'correctness' by means of deductive reasoning.
- The *technocratic* paradigm, promulgated mainly by software engineers, defines computing as an engineering discipline, treats programmes as mere data, and seeks probable, a posteriori knowledge about their reliability empirically using testing suites.
- The *scientific* paradigm, prevalent in the branches of artificial intelligence, defines computer science as a natural (empirical) science, takes programmes to be entities on a par with mental processes, and seeks a priori and a posteriori knowledge about them by combining formal deduction and scientific experimentation.

These paradigms provide a philosophical basis for the study of the methodology, ontology and epistemology, of computing, by treating them as either a branch of mathematics, or an engineering discipline, or a natural science.

We argue that both the rationalist paradigm and the scientific paradigm are inimical to creativity. This is because mathematics and science are 'rigid' approaches which require the identification of truth and the elimination of ambiguity. The technocratic paradigm, on the other hand, is flexible or 'not rigid' and hence creative. Given that there are three main elements in computing: hardware, software and communications, we observe that both hardware development and communications development follow mathematical and physical laws, leaving little room for creativity. Software, by contrast, has always involved some degree of creativity. Creative Computing will aim to develop and increase this creative component.

This delimitation of the field of Creative Computing is however not restrictive, because there are many different disciplinary perspectives from which to approach the topic. A few examples will illustrate the point.

An Engineering perspective would see Creative Computing as an engineering artefact, designed, tested and deployed using engineering methods, relying heavily on testing and inspection for validation. Selinger (2004) listed a number of creative engineers and their famous inventions, such as: Lonnie Johnson (SuperSoaker), Hedy Lamarr (spread-spectrum telecommunications), Shuji Nakamura (gallium nitride-based optoelectronics), Linus Torvalds (Linux), Randice-Lisa Altschul (disposable cellphone), Steve Wozniak (Apple II), Dean Kamen (Segway), Tim Berners-Lee (World Wide Web).

A Mathematical perspective would see Creative Computing as a theory which can be analysed for consistency and then refined into a more specialised theory.

A Cognitive Science perspective would see Creative Computing as a non-human agent, with its own personality and behaviour, defined by its past history and structural makeup.

A Social perspective would see Creative Computing as a social structure of software agents, who communicate, negotiate, collaborate and cooperate to fulfil their objectives.

All these perspectives (and there may well be others from Psychology, from Neuroscience, from Art, and so on) have something to contribute to the formation of our understanding of Creative Computing. In great measure, it will be the combination of these perspectives into transdisciplinary methodologies that will offer the greatest potential for progress in the field. Part of the aim of Creative Computing is to overcome the disciplinary boundaries that traditionally demarcate these approaches.

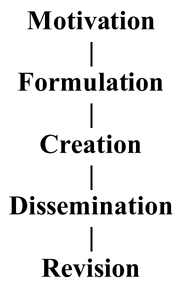
4 Comparison of creativity in music composition and software engineering

The first step towards understanding how Creative Computing may be implemented is to compare an undoubtedly creative process (in this case, music composition) with the process of software engineering. The nature of the creative process in art is famously elusive and subject to numerous conditioning factors. Nor is it, of course, the only form of creativity. But artistic creativity and Creative Computing have a number of features in common, as well as some subtle, yet important, differences. These provide an indication of the nature of the challenge facing computing if it is to become creative. Let us compare, therefore, the elements of the process of artistic creation for purposes of comparison with those of Creative Computing. In this instance, we will consider musical creation, but there are enough general similarities with other art forms for this to stand for all.

4.1 Creativity in music composition

Musical creativity consists of several interacting layers of activity (see Figure 1). Note that these will vary in weight and significance, depending on the project. Each is subject to numerous variables which will also change over time as the creative process unfolds. There are both external and internal factors that may condition a work's creation. In other words, the 'user requirements' may be dictated by external clients (commissioners, producers, performers, and so on) or by the composer him or herself. Note also that 'composer' here refers to the originator of the work, but this model is not constrained to the traditional authorial figure: an improviser (as in jazz) may be a composer, engaged in real-time composition. The principle even extends as far as an interpretative performer, whether this be a human being or a computational device or other machine. To some extent, all of these are working through the layers outlined below.

Figure 1 Musical creation: steps



Motivation in its simplest form comprises the question ‘Why compose?’ The possible answers are many and varied, ranging from the personal (inspiration, reaction to something, ‘just because...’, etc.) to the external (a commission, a request, a concept, etc.), or some combination of these. For a fuller discussion, see Hugill (2012).

Formulation comprises the early decisions about the composition. These may be subject to later revision as the process of creation unfolds. Typical early decisions cover both parametric and formal concerns. So, for example, the composer might decide at this stage on the instrumentation, duration, context, and influences (or models) that will shape the work. He/she may decide whether to work with sounds or notes, or a combination of those, and hence what type of notation (if any) will be used. The issue of the likely performance or dissemination will also be a key factor, including such questions as which musicians or instruments are to be involved, the venue for performance, or the way in which the work will be received, and so on.

Having established these wider constraints, some formal issues will most likely arise. Will there be a predefined structure (such as a verse-chorus-verse-chorus song, for example) to the composition? Will it be written ‘top-down’ by filling in the structure with musical content, or will it be ‘bottom-up’, beginning with a musical idea which is then grown and developed, or, as very often happens, somewhere between the two? How will the musical material be organised? Is there some kind of compositional system or procedure at work such as a structuring principle or pitch organisation? If so, will this be audible in the finished work?

Finally, practical considerations will come to the fore: what are the right tools for the job? Which instruments, software, paper, musicians, to use for the creation of the composition (rather than the finished piece, although these might well end up being the same)?

These kinds of decisions may be taken in any order and with varying degrees of weight or time given to each. Many of the decisions may not be consciously taken at all, especially when the composer is working in a way which is very familiar to them or the result of genre conventions. Often, reducing the amount of time spent on such decision-making is seen as a way to greater fluency and professionalism in composition.

Creation is the part of the compositional process that most people would recognise as creative. This is where the music itself is actually made, and is essentially a recursive process of origination, contemplation, and decision-making, intercalated with distraction, abandonment, certainty and doubt. Typical decision-making chains go: imagine something – realise it – listen to it (either in head or as sound) – evaluate it (using critical judgement) – keep or discard. Each of these stages may take any amount of time. When composing quickly (freely and easily), the whole process is compressed, sometimes to the point of apparent invisibility. Improvisation is the most obvious example of this.

There is always room for rehabilitation of discarded material, either in actuality or as a memory, depending on circumstances. The mind tends to flick back and forth between ‘creative’ origination and ‘reflective’ criticism. Too much of either can be bad for the finished result – a balance must be achieved. Too much naked creativity tends to lead to incoherent results. Too much criticism leads to creative blockage and awkward production. In many cases, a simple ‘that works’ is sufficient, but on other occasions the judgements may be more considered: how does it work? Is there another way of making it fit? Maybe there is more work to do to make this ‘work?’ And so on. In many cases, the relationship with a model (i.e., something already heard, or a previous composition) is a crucial jumping-off point.

Composers tend to constantly juggle the relationship between the macro (formal structure) and the micro (moment to moment). There is also the issue of horizontal (i.e., linear) and vertical (i.e., overlaid) composition. Sometimes one layers up, linear fashion. Other times one works vertically and creates textures. In many cases it is a bit of both. Deciding when a piece is finished is often very difficult. Some composers *never* really finish their works. Others just let them go after a time, or are forced to do so by circumstance.

It should also be said that there is something mysterious about the process of artistic creation. This mysteriousness persists despite the large amounts of neuroscientific and psychological research that has been done on creativity. It arises from the fact that some aspects of the creative process emanate from the subconscious mind. For many artists, accessing these subconscious processes is a creative goal in itself, and there is a natural wariness, even horror, of post-hoc rationalisations of creativity such as those described in this section of this article. There are numerous well-known examples of subconscious creativity, from Igor Stravinsky's famous statement "I was the vessel through which *Le Sacre du Printemps* passed" to Paul McCartney's oft-repeated story that he dreamt *Yesterday* and awoke convinced he had heard it somewhere before, until further investigation revealed that it was in fact original.

Dissemination in music takes many forms, of which the most familiar are the concert performance, the broadcast and the issue of a recording. These are points of actualisation, during which, often for the first time, it is possible for the artist to listen more objectively. People's reactions and critical comments can be formative at this stage. One must take a view whether to accept or ignore them, and to consider whether revisions might be made. The music now has a 'life of its own'.

Revision is optional, but quite common. Many compositions have been substantially rewritten after the first performance. On the other hand, it can be (in fact often is) best just to leave them to make their own way in the world.

4.2 Creativity in software engineering

Let us begin with the astonishing similarities between the process of musical creation described above and the process of software engineering. Figure 2 describes the software development process, whose layers closely echo those of musical creation depicted in Figure 1:

Figure 2 Software development: steps

Requirements Engineering
|
System Design
|
Coding
|
Operation
|
Evolution

We may firstly observe that, to be truly creative, the software must be able to solve the problem of human creativity in all its manifestations. The aspects of creativity that are most highly valued (Boden, 1998) are the ones that the software must be capable of exhibiting. This will inevitably lead to complexity, with the involvement of multiple disciplines and multiple users, all with different demands and constantly varying user requirements. There will be a high level of wastage and entropy in any creative process. Repetition and variation present specific problems. Finally, there is the problem of verification: how may we measure increased creativity, and how do we evaluate its quality?

The Guide to the Software Engineering Body of Knowledge (SWEBOK) (Abran and Moore, 2004), for which there is no equivalent in music composition, identifies the following steps in software development, which we here annotate with observations about the specific needs of Creative Computing.

Requirements engineering is widely used to denote the systematic handling of requirements. The SWEBOK states: "Software requirements should be stated as clearly and as unambiguously as possible, and, where appropriate, quantitatively. It is important to avoid vague and unverifiable requirements which depend for their interpretation on subjective judgment ("the software shall be reliable"; "the software shall be user-friendly")". This is where we encounter the first significant challenge to established orthodoxies. Creativity, far from stating its requirements clearly and unambiguously, actually often requires vague and unclear statements. Furthermore, these are invariably qualitative. Examples: "I want something that works", "I want something that seems to express <emotion>", "I want something that breaks out of the norm".

User requirements include both functional and non-functional requirements. The functional requirements will vary enormously depending on the forms of creativity being engineered. There is no single functional requirement for creativity per se. Non-functional requirements is, once again, user-defined on the fly. However, there is a degree of compromise: users cannot spend their time constantly re-defining their requirements. This in itself would become a barrier to creativity. Instead, the software automation must be able to deduce to redefinitions of requirements over time, based on probabilistic principles.

There is a paradox here. While the SWEBOK states that goals are often 'vaguely formulated', in fact creative goals are often very precisely formulated. For example: "I want to write a string quartet lasting 10 minutes for the X Quartet to perform in a concert of mixed classical and new work on such and such a day. Furthermore, the X Quartet has a reputation for playing microtonal music, so I want to write in a microtonal way, but not using the familiar clichés of such music from Haba onwards. I want to develop my own style in this music, which extends my previous work. And I want the piece to reflect my feelings about a recent tragedy". We could go even further in detailing these particulars. An alternative, 'bottom-up' set of goals might be: "I want to take the following set of timbres, pitches and rhythms and create a 10 minute string quartet that builds over time a structure that has an underlying sense of unity, etc. etc." It is the creative process itself which is much less precise.

The software engineers must have knowledge of the creative domain, but for this to be truly effective the creative people must also have knowledge of the software engineering. The software engineer needs to 'identify, represent, and manage the 'viewpoints' of many different types of stakeholders' rather more in the creative sector than in traditional industries. The operational and organisational environments are also crucial, as per the SWEBOK. However, we may say that the creative environment is also a conceptual environment, in a way that a call centre, for example, is not.

Our observations are as follows. For Creative Computing the software must be constantly rewritten since the parameter specifications will constantly vary. The parameters of process requirements will also be constantly rewritten by the users. Clearly, Creative Computing cannot be restricted to a single platform. The main system requirement, therefore, is cross-compatibility. User requirements are specifically ruled out of this definition by the SWEBOK, but they are the crucial aspect for Creative Computing.

The elicitation techniques given in the SWEBOK are all appropriate to Creative Computing requirements. The SWEBOKs model clearly implies a fixed development/use sequence. In Creative Computing, however, the process model should be continuous and never-ending, since usage will only continue to develop new life cycles and processes. The profile of the Process Actors is typical, except that the customers, users and markets are in the creative sector. Process Quality and Improvement are practical matters that will be defined by each project. This is a highly important aspect of the challenges facing Creative Computing and relies on effective transdisciplinary collaboration between the creative sector and the software engineers. This collaboration will itself throw up many ontological and semantic issues.

System design (Abran, 2004) is at once "the process of defining the architecture, components, interfaces, and other characteristics of a system or component" and "the result of [that] process". Viewed as a process, system design is the activity in which software requirements are analysed in order to produce a description of the software's internal structure that will serve as the basis for its construction. It must also describe the components at a level of detail that enable their construction. System design plays an important role in developing software: it allows software engineers to produce various models that form a kind of blueprint of the solution to be implemented. We can use the resulting models to plan the subsequent development activities, in addition to using them as input and the starting point of construction and testing.

System design consists of two activities that fit between software requirements analysis and software construction:

- software architectural design (sometimes called top-level design): describing software's top-level structure and organisation and identifying the various components
- software detailed design: describing each component sufficiently to allow for its construction.

Software construction (Abran, 2004), or coding, refers to the detailed creation of working, meaningful software through a combination of coding, verification, unit testing, integration testing, and debugging. The software construction process itself involves significant software design and test activity. It also uses the output of design and provides one of the inputs to testing, both design and testing being the activities. Software construction typically produces the highest volume of configuration items that need to be managed in a software project (source files, content, test cases, and so on).

The *Operation* stage of a system just shows that the system is running, serving the purpose for which it was constructed. Software development efforts result in the delivery of a software product which satisfies user requirements. Accordingly, the software product must change or evolve. Once in operation, defects are uncovered, operating environments change, and new user requirements surface.

Software maintenance (Abran, 2004), or *Evolution*, is the totality of activities required to provide cost-effective support to software. Activities are performed during the pre-delivery stage, as well as during the post-delivery stage. Pre-delivery activities include planning for post-delivery operations, for maintainability, and for logistics determination for transition activities. Post-delivery activities include software modification, training, and operating or interfacing to a help desk. The Evolution phase of the life cycle formally begins following a warranty period or post-implementation support delivery, but in fact maintenance activities occur much earlier. Software maintenance is an integral part of a software life cycle. However, it has not, historically, received the same degree of attention as the other phases.

Software development has historically had a much higher profile than software maintenance in most organisations. This is now changing, as organisations strive to squeeze the most out of their software development investment by keeping software operating as long as possible. Concerns about the Year 2000 (Y2K) rollover focused significant attention on the software maintenance phase, and the Open Source paradigm has brought further attention to the issue of maintaining software artefacts developed by others.

4.3 Crossing levels of abstraction

Given its importance to software engineering as well as computing, it might be assumed that abstraction is a well understood concept. On closer inspection that turns out not to be the case. In software engineering, abstraction is a cornerstone of the development process and is habitually used in a way that resembles mathematical concepts. This masks a more challenging reality, that, as Colburn and Shute (2007) propose: the distinction between abstraction in mathematics and abstraction in computing lies in the fact that in mathematics abstraction is *information neglect* whereas in computing it is *information hiding* (Parnas, 1976). That is, abstractions in mathematics ignore what is judged to be irrelevant (e.g., the colour of similar triangles). By contrast, in computing, any details that are ignored at one level of abstraction (e.g., Java programmers need not worry about the precise location in memory associated with a particular variable) must not be ignored by one of the lower levels (e.g., the virtual machine handles all memory allocations). Nevertheless, the concept of an abstraction is fundamental to software development because this is probably the best way to decompose a complex task into subtasks that may be effectively understood by human beings.

When we consider artistic creation, on the other hand, the word ‘abstract’ takes on a different meaning. In painting, for example, it refers to non-figurative imagery, that is to say iconography that does not appear to depict anything beyond itself. In music, the meaning is somewhat similar: sounds that have no meaning beyond themselves heard as (musical) *sounds.* This stands in contradistinction to what is often called ‘programme music’, which implies some kind of associated narrative. In some cases this is very explicit ‘storytelling’, in other cases it is more of musical argument. In both cases, it tends to lead to the idea that music might be a language. It is beyond the scope of this paper to discuss that idea, but suffice to say it has been very vigorously contested (see, for example, Dahlhaus, 1991).

5 Proposed framework for Creative Computing challenges

A new framework for tackling Creative Computing challenges must address the following questions:

- What are the reasons for having challenges in Creative Computing?
- What are the types of Creative Computing?
- What are possible mechanisms of developing Creative Computing?
- What is a suitable research method for studying Creative Computing?

5.1 Reasons for challenges in Creative Computing

In Creative Computing, as in software engineering, a significant portion of the process takes place in an abstract space between the concrete realities of user requirements (high level) and machine processes (low level). This space consists of abstract models and temporarily concretised representations. Since the overall process is driven by concrete realities, there are various constraints upon these models and representations that condition their nature. We are theorising that there are layers of abstract reality and that the motions between these layers is crucial to the nature of the completed outcome. We may call this the creative process. It takes place in conception rather than reality. The representations may in themselves be concretised, but they are only evidence of the existence of a conceptual process and have no value beyond assisting that process.

We propose to use four layers of abstraction, which correspond to the layers already identified in the creative process and software engineering. Each of these proceeds towards concretisation, as shown in Table 1:

Table 1 Layers of abstraction

Artistic creation	*Software engineering*	*Layer of abstraction*
Motivation	User requirements	Abstract
Formulation	System design	Less abstract
Creation	Coding	Less concrete
Dissemination/revision	Operation/evolution	Concrete

Each of these layers may be an action or an end result or even a combination of the two. In order to distinguish the Creative Computing model from the existing models above, and to suggest the synthesis between artistic and software creation, we propose the following four layers:

Figure 3 Creative computing: steps

Motivation (digitised thinking)
|
Ideation (design sketch)
|
Implementation (creative system)
|
Operation (effect of system/revision)

5.2 Types of Creative Computing

We have identified three interrelated types of Creative Computing, based on the properties of creativity (Mayer, 1999), as follows:

- creative development of a computing product
- development of a Creative Computing product
- development of a computing environment to support creativity.

Creative development of a computing product refers to the way in which people work in the process of software development. This focus on people enables the deployment of creative methods. In order to develop software creatively, people focus on Requirements Engineering, Software Design, Software Construction, Software Testing and Software Maintenance (SWEBOK).

Development of a Creative Computing product, by contrast, focuses upon the product itself. The aim is to improve creativity and expressiveness in the software itself. Software Quality is therefore the key indicator of this type.

The development of a computing environment to support creativity requires a focus on Software Configuration Management, Software Engineering Management, Software Engineering Process and Software Engineering Tools and Methods. The overall goal is to create a general purpose platform for computing products that enables the same focusing on creative processes as described earlier in this paper.

5.3 Mechanisms for Creative Computing

Boden (1998) identified three kinds of creativities. *Combinational* creativity involves an unfamiliar combination of familiar ideas, and requires a rich store of knowledge. The results are only valued if some link between them is perceived. *Exploratory* creativity explores within an established conceptual space. This is more likely to arise from a thorough and persistent search of a well-understood space. *Transformational* creativity, on the other hand, deliberately transforms a conceptual space. Transformational creativity should involve the rejection of some of the constraints that define this space and some of the assumptions that define the problem itself.

These three kinds provide the theoretical basis of the mechanisms for Creative Computing. The practical realities of their application must be worked out in different applications and circumstances, usually on a case-by-case basis.

5.4 Research methods for Creative Computing

Since Creative Computing is a new field of study, a suitable research method is needed. An initial review has been conducted to examine whether any of the existing research methods may be suitable (Seaman, 1999; Somekh and Lewin, 2006). The problem with these methods is their discipline-specificity. Creative Computing is inherently transdisciplinary, which renders most established methods unsuitable. Even the pragmatic approach of mixed methods research presents some problems in this field.

As we saw in Sections 4.1 and 4.2 above, a creative process has its own distinguishing features. It will involve an endlessly fluctuating mix of divergent and convergent thinking. A suitable research method should take these features into consideration. To give an example of one such methodology, here is the plan of a project currently under way within our research group:

a review the work done on creativity in a number of areas, i.e., natural science, art, humanity, business, social science and software.

b identify the key activities that constitute the creativity in this context

c analyse the processes of creation involved

d propose approaches to supporting creative activities and processes by software development

e design and implement a software system for one of the above approaches

f experiment with this system and propose a framework.

This method straddles artistic or 'creative' research and computer science methods. The combination of the two results in an approach that is flexible and involves a degree of creativity on the part of the researchers.

5.5 Software standards and Creative Computing

What are the software standards for Creative Computing? It is clear that a conventional approach to setting standards will not do in this case, because creativity itself is inherently not standardisable. Indeed, it may be argued that the first standard of Creative Computing is that it resists standardisation.

We would similarly argue that another standard is perpetual novelty. Software development that does not innovate, but rather settles for a known standard, is insufficiently creative. The following of traditional standards has led to software that fails to meet the needs of the users. This is because the standards themselves ossify and embed limitation.

A third standard is therefore defined by user interaction. We call this a 'constructive standard' and it is subject to continuous re-evaluation by the community of users. Since the standard is to encourage divergent and creative thinking, which is inherently subjective, then the measurement of the standard must be done subjectively and over time by the users. Writing this standardisation into the software itself should become standard.

Finally, we would argue that standards in Creative Computing should be *Combinational*, *Exploratory* and/or *Transformational.* In other words, they should reflect the mechanisms of creativity discussed earlier. Software may be measured using these three criteria in the search for meaningful standards of creativity.

6 What to discuss in the *International Journal of Creating Computing* (*IJCRC*)

IJCRC will encourage discussions of all issues related to Creative Computing and the components of the proposed framework outlined above. In addition, we have a number of particular areas of interest:

6.1 Transdisciplinary semantics

Computing scientists use words which are reassuringly familiar to artists and creative people. Terms like 'entity', 'ontology', 'semantics', and so on, are well understood by the humanities. However, the way computer scientists use these terms, and the highly rigorous ways in which they are implemented in software engineering, belies their typical 'fuzziness'. Attempts to address this need for ambiguity, such as 'fuzzy logic', generally seem to serve to create even more precision. Yet, both sides of the art/science divide seem to accept this situation as inevitable. How may we develop a metalanguage that better translates the inherent tensions between the two poles? The field needs to evolve towards a more fruitful transdisciplinary way of working. Users of software must be involved in the design process if this is to be achieved. Theorists and users must work together to achieve a synthetic understanding.

6.2 Novel design

How may we build subjective ambiguities into the software development process in order to counter the prevailing tendency towards ever greater neatness? In order to do this, computer science must analyse (or learn the lessons from existing analyses of) creativity more effectively. Novel design methodologies need to be designed to enable effective Creative Computing

6.3 Adaptive software

How may we create systems and software that recreates and adapts itself in real time? Simply offering a 'fixed' solution to the creative 'problem' is inadequate to the task of Creative Computing. Instead, software must be capable of permanent and dynamic evolution as the creative process unfolds.

7 Concluding remarks

Studies of computing have heretofore tended to concentrate upon on a single stream, such as Artificial Intelligence or Mathematics. It is not hard to see that, both currently and in the future, computing needs to deal with ever more complex applications and hence needs a framework under a new philosophy. Creativity can be the catalyst for such new developments in computing and especially in software. For that reason, this journal will concentrate on the transdisciplinary aspects of computing in terms of creativity.

References

Abran, A. and Moore, J.W. (2004) *Guide to the Software Engineering Body of Knowledge (SWEBOK)*, IEEE Computer Society Press, Los Alamitos, California.

Boden, M.A. (1998) 'Creativity and artificial intelligence', *Artificial Intelligence*, August, Vol. 103, Nos. 1–2, pp.347–356.

Boden, M.A. (2003) *The Creative Mind: Myths and Mechanisms*, 2nd ed., Routledge, London.

Broy, M. (2006) *The 'Grand Challenge' in Informatics: Engineering Software-Intensive Systems*, IEEE Computer.

Clarke, L.A., Osterweil, L.J., Perry, D. and Taylor, R. (1999) 'Software engineering research: challenges, successes, and opportunities', *NFS Software Strategies Workshop*, University of Southern California, August.

Colburn, T. and Shute, G. (2007) 'Abstraction in computer science', *Minds and Machines*, Vol. 17, No. 2, pp.169–184, Springer.

Dahlhaus, C. (1991) *The Idea of Absolute Music*, University of Chicago Press, Chicago.

Eden, A.H. (2007) 'Three paradigms of computer science', *Journal of Minds and Machines Archive*, Vol. 17, No. 2, pp.135–167, Kluwer Academic Publishers Hingham, MA, USA.

Finkelstein, A. (2011) *10 Open Challenges in Software Engineering: The Strachey Lectures in Computing Science*, February, Computer Science Department, Oxford University.

Hugill, A. (2012) *The Digital Musician*, Routledge, New York.

Hugill, A., Yang, H., Raczinski, F. and Sawle, J. (2013) 'The pataphysics of creativity: developing a tool for creative search', *Digital Creativity*, Vol. 24, No. 3, (Accepted, to be published September 2013).

Keane, M. (2007) *Created in China: The Great New Leap Forward*, Routledge, New York.

Mayer, R.E. (1999) *Fifty Years of Creativity Research: In Handbook of Creativity*, edited by R.J. Sternberg, pp.449–460, Cambridge University Press, Cambridge.

Parnas, D.L. (1976) 'On the design and development of program families', *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 1, pp.1–9.

Ritchie, G. (2007) 'Some empirical criteria for attributing creativity to a computer program', *Minds and Machines*, Vol. 17, No. 1, pp.67–99.

Seaman, C.B. (1999) 'Qualitative methods in empirical studies of software engineering', *IEEE Transaction on Software Engineering*, Vol. 25, No. 4, pp.557–572.

Selinger, C. (2004) 'The creative engineer – what can you do to spark new ideas?', *IEEE Spectrum*, August, pp.47–49.

Somekh, B. and Lewin, C. (2006) *Research Methods in the Social Science*, Sage Publications, London.