Journeys in Non-Classical Computation

A Grand Challenge for Computing Research

Susan Stepney and John A. Clark, University of York
Colin Johnson, University of Kent
Derek Partridge, University of Exeter
Robert E. Smith, University of the West of England

1 The Challenge

A *gateway event* (a term coined by Murray Gell-Mann) is a change to a system that leads to the possibility of huge increases in kinds and levels of complexity. It opens up a whole new kind of phase space to the system's dynamics. Gateway events during evolution of life on earth include the appearance of eukaryotes (organisms with a cell nucleus), an oxygen atmosphere, multi-celled organisms, and grass. Gateway events during the development of mathematics include each invention of a new class of numbers (negative, irrational, imaginary, ...), and dropping Euclid's parallel postulate.

A gateway event produces a profound and fundamental change to the system: once through the gateway, life is never the same again. We are currently poised on the threshold of a significant gateway event in computation: that of breaking free from many of our current "classical computational" assumptions. The Grand Challenge for computer science is

to journey through the gateway event obtained by breaking our current classical computational assumptions, and thereby develop a mature science of Non-Classical Computation

2 Journeys versus Goals

To travel hopefully is a better thing than to arrive.
– Robert Louis Stevenson, "El Dorado", 1878.

Many Grand Challenges are cast in terms of *goals*, of end points: "achieving the goal, before this decade is out, of landing a man on the moon and returning him safely to earth", mapping the human genome, proving whether P = NP or not. We believe that a goal is not the best metaphor to use for this particular Grand Challenge, however, and prefer that of a *journey*.

The metaphor of a journey emphasises the importance of the entire process, rather than emphasising the end point. In the 17th and 18th centuries it was traditional for certain sections of "polite society" to go on "a Grand Tour of Europe", spending several years broadening their horizons: the experience of the entire journey was important. And in the Journey of Life, death is certainly not the goal! Indeed, an open journey, passing through gateway events, exploring new lands with ever expanding horizons, need not have an end point.

A journey of a thousand miles begins with a single step.
– Lao Tzu, *Tao Te Ching*, Chapter 64, ~600 B.C.

Journeys and goals have rather different properties. A goal is a fixed target, and influences the route taken to it. With an open journey of exploration, however, it is not possible to predict what will happen: the purpose of the journey is discovery, and the discoveries along the journey suggest new directions to take. One can suggest starting steps, and some intermediate *way points*, but not the detailed progress, and certainly not the end result.

Thinking of the Non-Classical Computation Challenge in terms of a journey, or rather several journeys, of exploration, we suggest some early way points that appear sensible to aim for. But we emphasise that these *are* early points, that we spy today as we peer through the gateway. As the community's journey progresses, new way points will heave into view, and we can alter our course to encounter these as appropriate. Other members of the community may spy other way points, and are encouraged to explore these, too.

The Road goes ever on and on.
– J. R. R. Tolkien, *The Lord of the Rings*, 1954.

3 Background to the Challenge

Modern-day computing has radically changed our lives. It is an extraordinary success story. However, there is a growing appreciation that what passes as *classical computing* is an extremely small subset of all computational possibilities.

In many avenues of life, we often create unnecessary limitations. Perhaps the most invidious of these are the assumptions we make, particularly the implicit ones. We need to distinguish the true **this has to be the case** from the merely **this has always been the case.** Discoveries may emerge when what was considered an instance of the former is found to be an instance of the latter. Thus, for example, dropping Euclid's parallel postulate gave rise to non-Euclidean geometry. We wish to encourage similar revolt (with good academic motives) against the assumptions of classical computing. So we identify several features that define classical computing, but that may not necessarily be true in all computing paradigms, and we encourage the community to drop, invert, or otherwise perturb these in whatever ways seem interesting. Our brochure of reality-based journeys is a start.

Many computational approaches seek inspiration in reality (mainly biology and physics), or seek to exploit features of reality. These **reality-based computing** approaches hold great promise. Often, nature does it better, or at the very least differently and interestingly. Examining how the real world solves its computational problems provides inspirations for novel algorithms (such as genetic algorithms or artificial immune systems), for novel views of what constitutes a computation (such as complex adaptive systems, and self-organising networks), and for novel computational paradigms (such as quantum computing).

When it comes to the science of computation there is a gulf between the maturity of classical computing and that of the emerging non-classical paradigms. For classical computing, intellectual investment over many years is turning craft into science. To fully exploit emerging non-classical computational approaches we must seek for them such rigour and engineering discipline as is possible. What that science will look like is currently unclear, and we propose here to encourage exploration. The community must venture on its journeys of exploration, and report back its discoveries.

The development of a science of non-classical computing is a challenge indeed.

4 Six traditional paradigms to disbelieve before breakfast

We outline some classical computation assumptions, and ways they are being challenged by researchers in different fields. (Some of the categories arguably overlap. Later sections discuss alternatives in more detail.)

It ain't necessarily so.
– George Gershwin, *Porgy and Bess*, 1934

1. The Turing paradigm

classical physics: information can be can be freely copied, information is local, states have particular values. *Rather*, at the quantum level information cannot be cloned, entanglement implies non-locality, and states may exist in superpositions.

atomicity: computation is discrete in time and space; there is a before state, an after state and an operation that transforms the former into the latter. *Rather*, the underlying implementation realises intermediate physical states.

infinite resources: Turing machines have infinite tape state, and zero power consumption. *Rather*, resources are always constrained.

substrate as implementation detail: the machine is logical, not physical. *Rather*, a physical implementation of one form or another is always required, and the particular choice has consequences.

universality is a good thing: one size of digital computer, one size of algorithm, fits all problems. *Rather*, a choice of implementation to match the problem, or hybrid solutions, can give more effective results.

closed and ergodic systems: the state space can be pre-determined. *Rather*, the progress of the computation opens up new regions of state space in a contingent manner.

2. The von Neumann paradigm

sequential program execution. *Rather*, parallel implementations already exist.

fetch-execute-store model of program execution. *Rather*, other architectures already exist, for example, neural nets, FPGAs.

the static program: the program stays put and the data comes to it. *Rather*, the data could stay put and the processing rove over it.

3. The output paradigm

a program is a black box: it is an oracle abstracted away from any internal structure. *Rather*, the trajectory taken by a computation can be as interesting, or more interesting, than the final result.

a program has a single well-defined output channel. *Rather*, other observations can be made of the physical system as it executes.

a program is a mathematical function: logically equivalent systems are indistinguishable. *Rather*, correlations of multiple outputs from different executions, or different systems, may be of interest.

4. The algorithmic paradigm

a program maps the initial input to the final output, ignoring the external world while it executes. *Rather*, many systems are ongoing adaptive processes, with inputs provided over time, whose values depend on interaction with the open unpredictable environment; identical inputs may provide different outputs, as the system learns and adapts to its history of interactions; there is no prespecified endpoint.

randomness is noise is bad: most computer science is deterministic. *Rather*, nature-inspired processes, in which randomness or chaos is essential, are known to work well.

the computer can be switched on and off: computations are bounded in time, outside which the computer does not need to be active. *Rather*, the computer may engage in a continuous interactive dialogue, with users and other computers.

5. The refinement paradigm

incremental transformational steps move a specification to an implementation that realises that specification. *Rather*, there may be a discontinuity between specification and implementation, for example, bio-inspired recognisers.

binary is good: answers are crisp yes/no, true/false, and provably correct. *Rather*, probabilistic, approximate, and fuzzy solutions can be just as useful, and more efficient.

a specification exists, either before the development and forms its basis, or at least after the development. *Rather*, the specification may be an emergent and changing property of the system, as the history of interaction with the environment grows.

emergence is undesired, because the specification captures everything required, and the refinement process is top-down. *Rather*, as systems grow more complex, this refinement paradigm is infeasible, and emergent properties become an important means of engineering desired behaviour.

6. The "computer as artefact" paradigm

computation is performed by artefacts: computation is not part of the real world. *Rather*, in some cases, nature "just does it", for example, optical Fourier transforms.

the hardware exists unchanged throughout the computation. *Rather*, new hardware can appear as the computation proceeds, for example, by the addition of new resources. Also, hardware can be "consumed", for example, a chemical computer consuming its initial reagents. In the extreme, nanites will construct the computer as part of the computation, and disassemble it at the end.

the computer must be on to work. *Rather*, recent quantum computation results suggest that you don't even need to "run" the computer to get a result!

Doubtless there are other classical paradigms that we accept almost without question. They too can be fruitfully disbelieved.

5 The Real World : breaking the Turing paradigm

5.1 Real World as its own computer

The universe works! It doesn't need to calculate, it just does it. We can take the *computational stance*, and view many physical, chemical and biological processes *as if* they were computations: the Principle of Least Action "computes" the shortest path for light and bodies in free fall; water "computes" its own level; evolution "computes" fitter organisms; DNA "computes" phenotypes; the immune system "computes" antigen recognition.

This natural computation can be more effective than a digital simulation. Gravitational stellar clusters do not "slow down" if more stars are added, despite the problem appearing to us to be $O(n^2)$. And as Feynman noted, the real world performs quantum mechanical computations exponentially faster than classical simulations can.

5.2 Real World as our computer

Taking the computational stance, we may exploit the way the world works to perform "computations" for us. We set up the situation so that the natural behaviour of the real world gives the desired result.

There are various forms of real world sorting and searching, for example. Centrifuges effectively exploit differences in density to separate mixtures of substances, a form of *gravitational sorting*, if you like. Vapours of a boiling mixture are richer in the components that have lower boiling points (and the residual mixture is richer in those that have higher boiling points); distillation exploits this to give a form of *thermal sorting*. Chromatography provides another physical/chemical means of separation. In junk yards, ferromagnetic objects can be separated out by the use of industrial strength magnets. Other kinds of computations exist: optics can be exploited to determine Fourier transforms.

Maggots perform the "computation" of eating dead flesh: historically, maggots were used to clean wounds, that is, to perform their computation in a context to benefit us. More recently, bacterial metabolisms have been altered to perform the "computation" of cleaning up pollution.

Access control computations abound. Suitably constructed shape is used to calculate whether the key inserted in a tumbler lock is the correct one. Physical interlocks are exploited for safety and practical reasons across many industries: it is impossible to insert a nozzle from a leaded petrol pump into the fuel tank of a unleaded petrol car.

5.3 Real World as analogue computer

Properties of the real world may be exploited in other ways. The "computations" of the "real world as our computer" are very direct. Often we are concerned with more abstract questions. Sometimes the physical world can be harnessed to provide results that we need: we may be able to set up the situation so that there is an *analogy* between the computation performed by the real world, and the result we want.

There is an age-old mechanism for finding the longest stick of spaghetti in an unruly pile, exploiting the physics of gravity and rigidity: we can use this to sort by setting up an analogy between spaghetti strand length and the quantity of interest. Mercury and alcohol thermometers use a physical means of computing temperature by fluid expansion: the analogy is between the length of the fluid column and the temperature. Millikan's calculation of the charge on an electron exploits relationships between velocity of falling oil drops, viscosity of air, the charge on those drops and the strength of surrounding electric fields.

Classical computing already exploits physics at the level of electron movements. But there are other ways of exploiting nature.

Analogue computing itself exploits the properties of electrical circuits as analogues of differential equations.

DNA computing encodes problems and solution as sequences of bases (strands) and seeks to exploit mechanisms such as strand splitting, recombination and reproduction to perform calculations of interest. This can result in vast parallelism, of the order of 10^{20} strands.

Quantum computing presents one of the most exciting developments for computer science in recent times (the CS community greatly thanks the physicists for this). It breaks out of the classical Turing paradigm. As its name suggests, it is based on quantum physics, and can perform computations that cannot be *effectively* implemented on a classical Turing machine.[1] It exploits interference, many worlds, entanglement and non-locality. Newer work still is further breaking out of the binary mind-set, with multiple-valued "qudits", and continuous variables. Research in quantum computing is mushrooming, and it is apparent that we are not yet in position to fully exploit the possibilities it offers. If only small quantum computers were to prove practical then uses could still be found for simulating various quantum phenomena. However, if larger computers prove possible we will find ourselves unprepared.

- Why are there so few distinct quantum algorithms? How can new ones be found?
- How do we discover new a quantum algorithms to solve a given problem? How do we use existing algorithms to solve new problems? How can we find the best algorithms to use given limited computational resources? More generally....
  - **What would a discipline of quantum software engineering look like?** (Appendix A addresses this question in more detail.)

---

1 Analogue (as in continuous) computing also breaks the Turing paradigm. But the real world is neither analogue nor classically discrete; it is quantum. So analogue computing might be dismissed as of theoretical interest only. However, the same dismissal might then be made of classically discrete (classical) computation! (The real world is also relativistic, but that paradigm has not been embraced by computation theory, yet.)

- How can quantum computers be harnessed most effectively as part of a hybrid computational approach?

5.4 Real World as Inspiration

Many important techniques in computer science have resulted from observing the real world. **Meta-heuristic search** techniques have drawn inspiration from physics (simulated annealing), biology (genetic algorithms, genetic programming), social networks (ant colony optimisation) and other domains. Neural networks (based very loosely on the way the brain works) have proven remarkably useful. L-systems (based on plant growth) have proved more generally applicable. More recently computational approaches inspired by immunology have arisen.

These have all proved remarkably successful, or look highly promising, yet the science underpinning their use comes nowhere near matching the science of classical computing. Given a raft of nature-inspired techniques we would like to get from problem to solution efficiently and effectively, and we would like to reason about the performance of the resulting systems. But this falls outside the classical refinement paradigm.

- **What would a science of non-classical refinement look like?** A science would allow us, for example, to reason confidently about the behaviour of neural networks in critical applications, to derive highly effective systems targeted at highly limited resources.

In the virtual worlds inside the computer, we are no longer constrained by the laws of nature. Our simulations can go beyond the precise way the real world works. For example, we can introduce novel evolutionary operators to our genetic algorithms, novel kinds of neurons to our neural nets, and even, as we come to understand the embracing concepts, novel kinds of complex adaptive systems themselves. The real world is our inspiration, not a restriction.

- **How can we use nature inspired computation to build "better than reality" systems?** What are the computational limits to what we can simulate?
- What is the best you can do given many components, each with highly restricted memory and processing ability?

6 Massive parallelism : breaking the von Neumann paradigm

Parallel processing (Cellular Automata, etc) and other non-classical architectures break out of the sequential, von Neumann, paradigm.[2]

Under classical computational assumptions, a parallel computation can be serialised, yet parallelism has its advantages.

Real-time response to the environment. The environment evolves at its own speed, and a single processor might not be able to keep pace. (Possibly the ultimate example of this will be the use of vast numbers of nanotechnological assemblers (*nanites*) to build macroscopic artefacts. A single nanite would take too long, by very many orders of magnitude.)

Better mapping of the computation to the problem structure. The real world is intrinsically parallel, and serialisation of its interactions to map the computational structure can be hard. Parallelism also permits colocation of each processor and the part of the environment with which it interacts most.

And once the classical computational assumptions are challenged, we can see that serialisation is not necessarily equivalent.

Fault tolerance. Computation requires physical implementation, and that implementation might fail. A parallel implementation can be engineered to continue working even though some subset of its processors have failed. A sequential implementation has only the one processor.

Interference/interaction between devices. Computation requires physical implementation, and those implementations have extra-logical properties, such as power consumption, or electromagnetic emissions, which may be interpreted as computations in their own right (see earlier). These properties may interfere when the devices are running in parallel, leading to effects not present in a serialised implementation. (Possibly the ultimate example of this is the exponentially large state space provided by the superposed parallel qubits in a quantum computer.)

The use of massive parallelism introduces new problems. The main one is the requirement for

[2] The fact that the sequential paradigm is named after von Neumann should not be taken to imply that von Neumann himself was an advocate of purely sequential computation; indeed, he was also one of the early pioneers of CAs.

decentralised control. It is just not possible to have a single centralised source exercising precise control over vast numbers of heterogeneous devices (this is merely a covert attempt to serialise the system). Part of this problem is tackled by the sister **Grand Challenges in Ubiquitous Systems**, and part is addressed in the later section on **open processes.**

7 In the eye of the beholder : breaking the output paradigm

The traditional paradigm of program execution is that of an abstract computation processing an input to produce an output. This input-output mapping is a logical property of the computation, and is all that is important: no intermediate states are of interest, the computation is independent of physical realisation, and different instances of the computation yield precisely the same results.

Computation, however, is in the eye of the beholder. Algorithms are implemented by physical devices; intermediate states exist, physical changes happen in the world, different devices are distinguishable. Any information that can be observed in this physical world may be used to enrich the perceived computation.

Logical Trajectory Observations. An executing algorithm follows a *trajectory* through the logical state space. (Caveat: this is a classical argument: intermediate *quantum* computational states may be in principle unobservable.) Typically, this trajectory is not observed (except possibly during debugging). This is shockingly wasteful: such logical information can be a computational resource in its own right. For example, during certain types of heuristic search the trajectory followed can give more information about a sought solution than the final "result" of the search itself.

- How can logical observations made during execution be used to give useful information?

Physical Trajectory Observations. An executing algorithm is accompanied by physical changes to the world: for example, it consumes trajectory-dependent power as it progresses, and can take trajectory-dependent time to complete. Such physical resource consumption can be observed and exploited as a computational resource, for example, to deduce features of the logical trajectory. (For example, some recent attacks on smart cards have observed the power consumption profile and data-dependent timing of internal operations to deduce secret key information.) Such physical observations provide a very powerful source of information, currently exploited mainly by attackers, but available for more general computational use.

- What physical observations are feasible, and correlated with logical trajectories?
- What new uses can be found for such physical observations?

Differential Observations. An executing algorithm is realised in a physical device. Physical devices have physical characteristics that can change depending on environmental conditions such as temperature, and that differ subtly across *logically* identical devices. (Indeed, much of the rationale for digitisation is the removal of these differences.) So one can make observations not merely of the output of a single execution, but of set of outputs from a family of executions, of multiple systems, of different but related systems. For example, if repeated executions of a search each get 90% of elements of a sought solution correct then repeated executions might be combined to give an overall solution.

- How can diversity of multiple computations be exploited?
- How should diversity be engineered? By repeated mutation of a source program? By embracing technologically diverse solution paradigms?

Higher-order Observations. These are observations not of the program execution itself, but of the execution of the program used to design (the program used to design...) the program.

8 Open processes : breaking the algorithmic paradigm

In the classical paradigm, the ultimate goal of a computation is reaching a fixed point: the final output, the "result" of the computation, after which we may switch off the computer. The majority of classical science is also based around the notion of **fixed-point equilibrium** and **ergodicity** (ergodicity is the property that the system has well defined spatial and temporal averages, because any state of the system will recur with non-zero probability).

Modern theories of physics consider systems that lack repetition and stability: they are **far from equilibrium** and **non-ergodic**. Perhaps the most obvious non-ergodic, far from equilibrium system is that of life itself, characterised by perpetual evolution (change). Most human problems are also best described in such terms; since computation is ultimately in service of such problems, the implications of non-ergodic, far from equilibrium physics must be considered in relationship to computing's future.

Consider the most basic of chaotic systems: the logistic process, parameterised by R.

$$x_{t+1} = Rx_t(1-x_t)$$

The behaviours of various logistic processes as a function of R are shown in Figure 1, where each point on the plot is a point on the attractor.

For values of $1 < R < 3$, these logistic processes have a fixed point attractor. For $R = 3$ they an attractor of period two. As we raise R, the attractor becomes period four, period eight, etc. This *period doubling* continues as we raise R, and the values of R where each doubling occurs get closer together. For $R > 3.569945671\ldots$ the logistic process's attractor goes through an infinite number of values (except for a few "islands" or order, of attractors with multiples of odd periods).

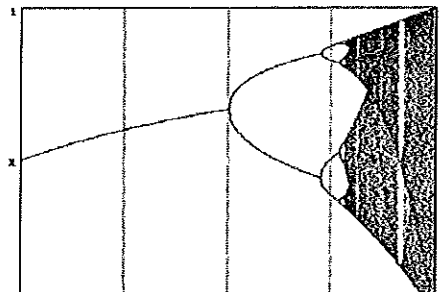

Figure 1: Points on the attractors of various logistic processes, versus the parameter R

There is a *phase transition* from order (the region of period doubling) to chaos ("random" behaviour). The phase transition point at $R = 3.569945671\ldots$ is the so-called *edge of chaos*.

Imagine taking measurements from a process whose underlying (continuous) dynamics are those of the logistic equation. Take very coarse measurements: say the process outputs symbol 1 if $x > 0.5$, and 0 otherwise, and take samples of length L bits. Construct an automaton machine that represents the process, for a given L. So the logistic processes generated by various values of R can be interpreted as a variety of *logistic machines*. There is a clear phase transition (a peak in the machine size versus the entropy of the bit sequence) as we move from the period doubling region to the chaotic region.

At the phase transition, the machine size versus the length of the sequence L, *expands without bound.* That is, at the edge of chaos, the logistic machine requires an infinite memory machine for accurate representation. There is a leap in the level of intrinsic computation going on in the logistic machine at the edge of chaos. (In terms of the Chomsky hierarchy, the machine has gone from the level of regular grammars to the level of context-free grammars.)

At the edge of chaos, the addition of new resources (computational or physical) can yield results that are neither redundant (as in the structured period doubling regime) nor random (as in the chaotic regime). Within the classical paradigm, such conditions would be anathema, indicating unceasing variety that never yields "the solution". But in life-like systems, there is simultaneously sustained order, and useful innovation. New matter can be brought into such systems and used in ways that are neither redundant nor random. In this setting, emergence of the unforeseen is a desirable property, rather disruptive noise.

Computing often attempts to exploit the biological paradigm: cellular automata, evolutionary computation, recurrent networks (autocatalytic, neural, genomic, immune system, ecological webs, ...), social insect and agent-based systems, DNA-computing, and nanite-systems that build themselves. However, in most of these cases, the implementations of such systems have been locked into themselves, closed, unable to take on new matter or information, thus unable to truly exploit emergence.

We should consider **open systems**, systems where new resources, and new kinds of resources can be added at any time, either by external agency, or by the actions of the system itself. These new resources can provide **gateway events**, that fundamentally alter the character of the system dynamics, by opening up new kinds of regions of phase space, and so allowing new possibilities. Computational systems are beginning to open themselves, to unceasing flows of information if not so much to new matter. The openness arises, for example, through human interactivity as a continuing dialogue between user and machine, through unbounded networks, through robotic systems with energy autonomy. As computers become ubiquitous, the importance of **open systems physics** to understanding computation becomes critical. The solutions we expect from people are ongoing processes, and this should be our expectation from computers too.

So some way points for the Grand Challenge are

- **Computation as a dynamical process.** What are the various attractors of a dynamical computation? How can we encourage the system to move to a "better" attractor? How can we map the route through intermediate attractors that it should take?
- **Computation at the edge of chaos.** What are it capabilities? How can we hold a system at the edge, far from equilibrium, to perform useful computatons? How can we make it *self organise* to the edge?
- **Open systems science.** What are the fundamental properties of open systems? How can we predict the effect of *interventions* (adding new things, or removing things) to the system? How can we understand the effect of a gateway event that opens up new kinds of regions of phase space to a computation?
- **Designed emergence.** How can we design (refine) open systems that have desired emergent properties? And do not have *un*desired emergent properties?

9 Maturity means hybrid solutions

Classical physics did not disappear when modern physics came along: rather its restrictions and domains of applicability were made explicit.

Similarly, non-classical computation will not supersede classical computation: it will augment and enrich it. And when a wide range of tools is available, it becomes possible to pick the best one, or the best combination, for each job. For example, it might be that using a quantum algorithm to reduce a search space, and then a meta-heuristic search to explore that, is more effective than using either algorithm alone.

- **Hybrid Solutions.** An array of novel traditional approaches to computation will become available. Can we create a general flexible conceptual framework that allows effective and efficient exploitation of hybrid approaches?

10 Exemplars – achievements of the journeys

The journey is the important thing. At various points in journey-space researches will alight to mark their way, leaving behind diary entries to which they may return at a later date. In common parlance these intermediate recordings may be regarded as "achievements". Opportunities are manifold. We expect sub-challenges in the sub-disciplines to be articulated separately. Indeed, two have already been prepared (see appendices):

- **Non-Classical Physics: Quantum Software Engineering**
- **Non-Classical refinement: Approximate Computation**

(And we have noted above the sister Ubiquitous Systems challenges.) Here we identify some further expected or possible achievements of the overall Grand Challenge in Non-Classical Computation.

- "pretty good solutions to currently intractable problems"
- quantum software engineering
- exploiting limited resources (lots of limited devices, ubiquity / having only a few qubits)
- a science of nature-inspired techniques
- understanding the edge of chaos and emergence
- many new applications become possible
- Nano-tech-ville
- hybrid solutions
- a "European Santa Fe" that complements the US's Santa Fe institute – plus a body of work that justifies it.

11 The Grand Challenge Criteria

It arises from scientific curiosity about the foundation, the nature or the limits of a scientific discipline. It arises from questioning the

assumptions of the classical paradigms, and aims at the creation of a new science.

It gives scope for engineering ambition to build something that has never been seen before. It aims to build a new science; the engineering opportunities will follow.

It will be obvious how far and when the challenge has been met (or not). It will never be met fully: it is an open journey, not a closed goal. The science will continue to mature, until itself overtaken by the next paradigm shift.

It has enthusiastic support from (almost) the entire research community, even those who do not participate and do not benefit from it. No. However, in the best tradition of paradigm shifts, the change will occur.

> ***An important scientific innovation rarely makes its way by gradually winning over and converting its opponents: it rarely happens that Saul becomes Paul. What does happen is that the opponents gradually die out, and that the growing generation is familiarised with the ideas from the beginning.***
> – Max Planck, *Scientific Autobiography*, 1949

It has international scope: participation would increase the research profile of a nation. This is a new fundamental area of computer science.

It is generally comprehensible, and captures the imagination of the general public, as well as the esteem of scientists in other disciplines. Much popular literature already exists in several of these areas, written by scientists in other disciplines (quantum computing, complexity, nanotech, ...), and so they and the general public are arguably already ahead of the CS community!

It was formulated long ago, and still stands. Its seeds have been around for a long time, but it has only recently become of obvious importance.

It promises to go beyond what is initially possible, and requires development of understanding, techniques and tools unknown at the start of the project. The structure of the Challenge mirrors the journey suggested by this criterion.

It calls for planned co-operation among identified research teams and communities. It is a multi-disciplinary Challenge, with contributions needed from a range of research specialities.

It encourages and benefits from competition among individuals and teams, with clear criteria on who is winning, or who has won. There need not be a single "winner". Diversity of solutions should be encouraged to be applicable to a range of application domains. Winners may emerge in particular application domains, as the strengths of the various techniques become clear.

It decomposes into identified intermediate research goals, whose achievement brings scientific or economic benefit, even if the project as a whole fails. There are several components to the Challenge that can be explored in parallel.

It will lead to radical paradigm shift, breaking free from the dead hand of legacy. Non-classical computing *is* a radical paradigm shift!

It is not likely to be met simply from commercially motivated evolutionary advance. Applications might be supported by industry, but it is unlikely that the development of the underlying science would be.

Appendix A: Non-Classical Physics: Quantum Software Engineering

This sub-challenge of Non-Classical Computation covers

the development of a mature discipline of Quantum Software Engineering

We wish to be ready to exploit the full potential of commercial quantum computer hardware, once it arrives, projected to be around 2020 (or, less optimistically, "20 years from now", whenever "now" is).

We might have to wait a while for commercial quantum computers, but when they arrive, Moore's law suggests they will grow in power *very quickly*. Doubling a classical computer's register length (roughly) doubles classical computing power, but adding just *one bit* to a quantum computer's register doubles quantum computing power. We need to be ready to exploit these devices once they appear. However, the majority of today's theory of computation, algorithms, programming languages, specification models, refinement calculi, and so on, is purely classical. The challenge is to build the corresponding languages, tools and techniques for quantum software engineering.

We need to raise the level of thinking about quantum programs. Today we reason about quantum programs predominantly at the level of quantum *gates*: imagine how far classical computing would have progressed if the only language we had to describe programs was that of AND and OR gates! Most importantly, we need a new paradigm (or paradigms) for thinking about quantum computations, to augment the existing classical declarative, functional, and imperative paradigms.

The whole of classical software engineering needs to be reworked and extended into the quantum domain.

Foundations

Much foundational work is still needed. We need further developments of the fundamentals of quantum computability: the **Universal Quantum Turing Machine.** We need to investigate **quantum algorithmic complexity:** time, space, "parallel universe space", and any other parameters of interest.

We have models of classical computation – von Neumann machines with fetch-execute-store, imperative, functional and logic languages, etc – that let us write and reason about classical programs without worrying about logic levels, transistors, gates, etc. In much the same way we need **metaphors and models of quantum computation,** that enable us design and reason about quantum algorithms without recourse to QM, unitary matrices, etc. Does Deutsch's many-worlds description provide the best programming metaphor, or are there better ones? Whatever the actual metaphors chosen, they must be formalised into new computational models.

We need theories and models of that weirdest quantum process of all: that of **quantum entanglement.** Two qubit entanglement is relatively well understood – but multi qubit entanglement, and qudit entanglement, are barely understood.

Languages and Compilers

We need to determine the **fundamental building blocks** of quantum programming: is there a simple extension of GCL? of classical logic languages? of classical assembly languages? is an entirely new paradigm needed?

We need to design suitable **assembly level** and **high level Q-languages** (analogues of classical imperative, declarative, and functional languages, at 3rd, 4th, 5th generation, and beyond). We need to design and build the corresponding **Q-compilers** for these languages.

We need to design and implement (initially, simulate) **new Q-algorithms** (beyond the current ones of Min Max, Shor's period finding algorithm used for factorization, and Grover's algorithm for DB searching). What classes of algorithms may be quantised? How may certain well-known classical algorithms be quantised?

We need to develop suitable **reasoning systems** and **refinement calculi** for these languages. (Even sequential composition is different in the quantum regime, due to the fundamental unobservability of the intermediate state.) Although higher level specifications may well abstract away from details of any underlying classical *versus* quantum implementation, there may be certain application-specific **quantum specification languages,** for example, for **quantum protocols.**

Methods and Tools

Before commercial quantum computers are available, we have to make do with simulations on classical machines. We need to implement powerful **quantum computer simulators,** in order to perform computational experiments and validate language and algorithm designs. (Computational resources for simulating quantum algorithms can be exponentially large. Something like a simulation engine of multiple FPGAs might be appropriate, to get the required massive parallelism.)

We need to discover what **higher level structuring techniques and architectures** are suitable for quantum software. In particular, can classical structuring (such as object-orientation, or component based software), be extended to incorporate Q-software? How can classical and quantum paradigms co-exist? (It seems likely that, at least to start with, most software will remain classical, with a "call-out" to quantum power as needed. But the development process needs to be able to handle such hybrid developments seamlessly.)

Given that quantum execution is *in principle* unobservable, we need to discover new **debugging and testing techniques** for these Q-languages.

We need to design ways of **visualising** Q-algorithm execution, as an aid to understanding, design, and implementation.

Novel Quantum possibilities

Quantum computing can do some things that cannot even be simulated by discrete deterministic classical computers. We need to extend quantum software engineering to encompass these new domains.

Quantum devices can produce **genuine random numbers;** classical digital simulations can produce only *pseudo*-random numbers. We need to investigate the differences this causes, if any. In the short term, will a quantum computer simulator need to be hooked up to a genuinely random number source? In the longer term, what new power, what new difficulties, might emerge as a result of genuine randomness?

Quantum **entanglement** offers many new possibilities, such as information teleportation. We need to understand how entanglement can be

applied to produce genuinely new algorithms, and new kinds of protocols.

Appendix B: Non-Classical Refinement: Approximate Computation

This sub-challenge of Non-Classical Computation is

to develop a science of approximate computation, and to derive from it a well-founded discipline for engineering approximate software

A radical departure from discrete correct/incorrect computation is required, a shift away from logics towards statistical foundations, such that meaningful estimates of 'confidence' emerge with each approximate result. This implies that probabilities play an integral part in computation throughout the process. The component probabilities and the eventual confidence estimates, if secured by large numbers (e.g. repeated sampling from a proposed distribution), imply a computational effort that is becoming increasingly feasible as a result of hardware advances as well as innovative developments in statistical modelling theory (e.g. reversible-jump Markov Chain Monte Carlo methods).

Classical computation versus approximations

The classical, discrete, view of computation has each step as either correct or incorrect, and the middle ground excluded. This naturally leads to formal logics as the dominant underpinning framework. The programmer devises the "formula", which is intended to be an exact solution to the problem; this symbol structure is translated into a machine executable form and the manipulations that the programmer envisaged are performed automatically, at high speed and with complete accuracy.

Consider the symbol structures being manipulated by a trained a multilayer perceptron (MLP), for example. These are not formulae composed of operators and variables that admit a ready mapping to the operations and parameters of the human conception of the problem. One consequence is that any adjustment to the function to be computed by an MLP involves complete retraining, because code-fixing is not an option. The "formulae" cannot reasonably be devised by a programmer; they must be automatically generated from data samples.

Typically, the output of an MLP classifier, a real-value, is arbitrarily thresholded to obtain a class label. This and other inherent weaknesses of an approximate classifier constructed with empirically determined (suboptimal) values for its parameters are widely acknowledged. *Ad hoc*-ery is rife in neural computing, but work on error-bars already points the way towards a well-founded science.

These innovative developments to move beyond the constraint of correct/incorrect results from hand-crafted formulae are but piecemeal strategies; they need to be woven into the basic fabric of a comprehensive model for approximate computation, not stitched-on to classical computation as useful extras, or mere curiosities.

How would the classical paradigm be shifted?

Taking the viewpoint that the computational task is an unknown (or intractable, see later) function, the computational goal is to approximate it in a way that holds the promise of reasonable optimality, but crucially associates a *meaningful estimate of confidence* with every output computed. In general terms, data-driven software development supplants specification-driven; computational tasks are viewed as data-defined rather than (abstract) specification-defined.

In detail, the approach might be through a survey, sampling by, say, Markov Chain Monte Carlo methods across a continuum of potentially viable models. By doing this within a Bayesian framework, rationalisable probabilities are attached to various elements throughout the computational process. The outcome is a weighted average across a range of modelling possibilities. It is a well-founded approximation whose validity emerges as a secure estimate from the computational processes employed. The infrastructure of the new paradigm seeks to avoid searching, comparing and selecting from amongst a discrete set of alternative models (and hence commitment to a specific model, or even discrete set of alternative models) by maintaining the range of potential models as a set of continuous parameters; probability theories, secured by large-number sampling, provide the over-arching framework.

A fundamental basis of continuity avoids the brittleness inherent in discrete, classical computation. Notice, for example, that the necessary discretisation of the real numbers that plagues classical computation is not similarly problematic for MLPs, despite their fundamental dependence upon the real continuum.

Initially at least, classical computation will provide the virtual machine upon which the approximate computations will run, but hardware innovations

coupled with the establishment of generally applicable approximation algorithms could change that dramatically. However, building the required confidence in a classically programmed virtual machine is not the same scale of problem as doing it individually for every piece of application software.

The initial challenge is to begin to establish the limits and the potential infrastructure of such a science of approximate computation. This includes major subdomains, such as a discipline of engineering approximate software. It also involves the identification and integration into a coherent framework of many activities that are currently pursued under a variety of labels, for example, statistical pattern recognition, some varieties of data mining, statistical data modelling, some technologies of inductive generalization or data-driven computation.

A science of approximate computation: when and where?

The new science of approximate computation will not oust the classical one; it will sit alongside it as a new weapon in an armoury of well-founded alternative computational techniques to be used when appropriate.

It will be appropriate to use whenever a computational task is defined more by samples of desired or observed behaviour than by an abstract specification. It will also be appropriate to use whenever the problem is well defined but computationally intractable, where the particular task is appropriate for approximate solutions, albeit with a 'confidence' measure attached; there is no prohibition on certainty emerging as an extreme of approximation.

Consider an illuminating extreme --- safety-critical software. Such systems would seem to absolutely require the classical strategy: they must be correct. However, the practical impossibility of this requirement leads to a slight relaxation: it is typically couched in terms of a very low failure/error rate, and the major component of the required assurances is extensive testing. The bulwark of statistical reasoning, as an integral part of the testing, is thus dragged in by the back door (as it were) – how much better to integrate it into the fabric of the computation from beginning to end, instead of tagging in on the end as a stopgap for verification failure?

Will 'programming' an approximation computer be more difficult than conventional programming? All we can say is it will be fundamentally different – for example, data analysis, selecting sampling strategies, rather than formula derivation. The 'programming' difficulties that confront the user of this new paradigm will be directly determined by how successful we are in formulating the fundamental model(s) of approximate computation.

Nothing in the above *requires* a new paradigm: any of the innovations envisaged could be realised within the scope of classical computation, as some already are. However, although a screwdriver can be used to open a tin, it is quicker, neater and generally preferable to use a well-designed tin opener for the task.